CXperf User's Guide

First Edition

B6323-96001
Customer Order Number B6323-90001
June 1998

Edition: First

Document Number: B6323-90001
Remarks: Released with HP CXperf V6.0, June, 1998.

Notice

© Copyright Hewlett-Packard Company 1998. All Rights Reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Contents

Preface ..... **xi**
System platforms xii
Notational conventions xiii
Associated Documents xv

1 Introduction **1**
Profiling methods 2
CXperf overview 3
Using CXperf 4
CXperf interfaces 5
GUI mode 5
Line mode 6
Batch mode 6
Graphical analysis 7
Summary Profile and Parallel Profile 7
Call Graph 8
Performance Reports 9
Metrics 9

2 Getting started **11**
Overview of a profiling session 12
Profiling a program in GUI mode 14
Compiling 14
Instrumenting 15
Executing 19
Analyzing 20
Profiling a program in line mode 22
Compiling 22
Instrumenting 24
Executing 27
Analyzing 28
Editing the command line 28

3 Preparing programs to profile **31**
Compiling 32
+pa and +pal options 33
Syntax 34
Compiling and linking in one step 35
Compiling and linking separately 35

Using CXoi to instrument object files and archive libraries 38
Syntax 38
Preparing for profiling 39
Instrumenting with CXoi 39
Linking the instrumented files 39
CXoi limitations 40

4 Choosing Data 41

Introducing metrics 42
Metrics available on all architectures 43
Architecture-dependent metrics 44
Process events 44
Memory events 45
Data Cache Utilization 45
Data and Instruction TLB misses 46
Derived metrics 46
Using event metrics 47

Instrumenting 49
Instrumenting in GUI mode 49
Selecting routines and loops 49
Selecting loop nesting levels 53
Selecting metrics to collect 56
Instrumenting in line mode 58
Selecting routines and loops 58
Selecting loop nesting levels 61
Selecting metrics to collect 63

Preinstrumenting 65
Setting the environment 65
Performance Data Files (PDFs) 65
CXperf command line options 66
The PROFDIR environment variable 66
Preinstrumenting in GUI mode 67
Preinstrumenting in line mode 69

5 Profiling 73

Profiling strategy 74
Profiling intrusion 74
Minimizing intrusion 76
Routines that call uninstrumented routines 77

Profiling MPI and PVM applications 79
Generating PDFs 79
Using CXmerge 80
Syntax 80
Analyzing merged data 81

Using Performance Data Files (PDFs) 83
Invoking CXperf with a PDF 83
Changing PDFs during a CXperf session 83
Batch mode. 85
Using a command file. 85
Command file input using the -x option 85
Argument input using the -e option. 86
Using a script 86
6 Analyzing 89
Analysis Page. 90
Toolbar 92
Configuration options. 95
Region 95
Metric. 98
Graphical Analysis 100
Accessing profiling data 100
Summary Profile. 102
Region Detail 103
Source Window 105
Parallel Profile 106
Call Graph 108
Text Reports 110
Accessing profiling data in GUI mode. 110
Accessing profiling data in line mode 112
Using `analyze` 113
Using `set pdf` 115
Using `set visibility`. 116
Using `list` 116
Using `list selectable`. 117
Report fields 119
Summary and Parallel Reports 122
Routine Performance Analysis 125
Loop Performance Analysis 125
Parallel Loop Performance Analysis 127
Call Graph Report 128
Line Mode Report 131
Using `analyze` 131
Using `set pdf` and `set visibility` 131
Viewing source in line mode 133
Glossary 135
Index. 139

Figures

Figure 1 Call Graph 8
Figure 2 Profiling using CXperf. 12
Figure 3 Select regions to profile 16
Figure 4 Select loop nesting level to profile 17
Figure 5 Select Metrics and Call Graph 18
Figure 6 Execution Page 19
Figure 7 Analysis Page. 20
Figure 8 Compilation Page 32
Figure 9 Browse: Select a file. 33
Figure 10 Compiling and linking separately 36
Figure 11 Instrumentation Page 50
Figure 12 Instrumentation Page: Select Regions to Profile. 52
Figure 13 Instrumentation Page: Default Loop Nesting Level 54
Figure 14 Instrumentation Page: Select Fixed Loop Nesting Level 55
Figure 15 Instrumentation Page: Select Relative Loop Nesting Level 56
Figure 16 Instrumentation Page: Select Metrics to Collect 57
Figure 17 Instrumentation Page: Preinstrument Executable 67
Figure 18 Uninstrumented child processes. 77
Figure 19 Analysis Page. 91
Figure 20 Find Region dialog. 93
Figure 21 Save Profile dialog. 94
Figure 22 Analysis Page: Region 95
Figure 23 Sort Criteria dialog 96
Figure 24 Subset Selection dialog 97
Figure 25 Analysis Page: Metric 98
Figure 26 Select Metric 99
Figure 27 Data Source dialog. 99
Figure 28 File menu: Open File. 101
Figure 29 Summary Profile 102
Figure 30 Region Detail dialog 104
Figure 31 Source Window 105
Figure 32 Parallel Profile. 106
Figure 33 Call Graph 108
Figure 34 Summary Report 122
Figure 35 Parallel Report. 124
Figure 36 Call Graph Report 129
Figure 37 Line Mode Report 132

Tables

Table 1 Compilers 14
Table 2 Compile instructions 23
Table 3 `set events` options 26
Table 4 Editing the command line 29
Table 5 -tm <*architecture*>: valid values 66
Table 6 Intrusion for loop profiling 75
Table 7 Region configurations 95
Table 8 Metric configurations 98
Table 9 Profiling Status 121

Preface

This guide describes the CXperf Performance Analyzer, an interactive runtime performance analysis tool for programs compiled with HP ANSI C (c89), ANSI C++ (aCC), Fortran 90 (f90), and HP Parallel 32-bit Fortran 77 (f77) compilers. This guide helps you prepare your programs for profiling, run the programs, and analyze the resulting performance data.

The *CXperf Command Reference* supplements this guide with CXperf command information. You can access online help in CXperf to get help on the GUI and CXperf commands.

You should already have experience developing UNIX applications.

CXperf has a variety of features that help you assess performance of your applications. These features include:

- GUI, line, and batch mode operation
- Profiling routines, loops, and compiler-generated parallel loops
- Profiling MPI and PVM message passing applications
- Routine level profiling for object files and archive libraries created with PA_RISC targeting compilers
- Preinstrumentation for executable files
- Graphical and textual analysis for performance data
- Performance Data File analysis for files created on a different architecture

System platforms

CXperf supports the following HP-PARIC 7200 and PA-RISC 8200 hardware platforms:

- V-Class
- D-Class
- K-Class

CXperf version 6.0 runs under the HP-UX 11.0 operating system. You must have the HP-UX 11.0 Extension Pack, June 1998 (XR39/IPR9806) installed to run CXperf.

Notational conventions

This section describes notational conventions used in this book.

`bold monospace`	In command examples, **`bold monospace`** identifies input that must be typed exactly as shown.
`monospace`	In paragraph text, `monospace` identifies command names, system calls, and data structures and types. In command examples, `monospace` identifies command output, including error messages.
italic	In paragraph text, *italic* identifies titles of documents. In command syntax diagrams, *italic* identifies variables that you must provide. The following command example uses square brackets to indicate that the variable *output_file* is optional: `command` *input_file* [*output_file*]
Brackets ([ ])	In command examples, square brackets designate optional entries.

Curly brackets ({}), Pipe (|)	In command syntax diagrams, text surrounded by curly brackets indicates a choice. The choices available are shown inside the curly brackets and separated by the pipe sign (|). The following command example indicates that you can enter either `a` or `b`: `command {a | b}`
Horizontal ellipses (...)	In command examples, horizontal ellipses show repetition of the preceding items.
Keycap	**Keycap** indicates the keyboard keys you must press to execute the command example or user selectable buttons on the Graphical User interface.

NOTE A note highlights important supplemental information.

CAUTION **A caution highlights procedures or information necessary to avoid damage to equipment, damage to software, loss of data, or invalid test results.**

Associated Documents

Associated documents include:

- *CXperf Command Reference V1.0*
- CXperf Online Help
- *Parallel Programming Guide for HP-UX systems*
- *HP MPI User's Guide*

1 Introduction

This chapter provides introductory information about CXperf and performance analysis. You are introduced to profiling methods and CXperf's features. Topics covered include:

- Profiling methods
- CXperf overview
  - Using CXperf
  - CXperf interfaces
  - Graphical analysis
  - Performance Reports
  - Metrics

Profiling methods

The methods available to carry out performance analysis are independent of underlying hardware, and are categorized by how data is collected—event-based versus statistical sampling.

Most performance analysis tools, including CXperf, require special profiling options when you compile a program. The options instruct the compiler to create a special executable file containing information that the profiler uses to collect performance data.

Statistical sampling profilers sample a program's performance at measured intervals and average each routine's execution time. The `gprof` and `prof` utilities use statistical sampling.

Event-based methods measure a program's entire execution time and report the total time spent in individual routines and loops. CXperf is an event-based profiler.

Event-based profilers have advantages over statistical sampling profilers; they provide a greater variety of metrics and direct correlation to the source code. Event-based methods of profiling can become intrusive. Keep the level of intrusion to a minimum. Due to increased profiling time and intrusion, selecting to profile all region types, all loops, and all metrics during a single profiling session is not recommended.

CXperf overview

CXperf is an interactive runtime performance analysis tool for programs compiled with HP ANSI C (c89), ANSI C++ (aCC), Fortran 90 (f90), and HP Parallel 32-bit Fortran 77 (f77) compilers.

To profile Fortran 77 programs, you must use the HP Parallel 32-bit Fortran 77 compiler. CXperf version 6.0 does not support the standard HP Fortran 77 compiler.

CXperf profiles selected parts of a program, controls the program's execution, stores performance data in a performance data file (PDF), and displays performance information in reports and graphs. CXperf supports

- Profiling routines, loops, and compiler-generated parallel loops
- Routine level profiling for object files and archive libraries created with PA_RISC targeting compilers
- Displaying profiling information for
  - Entire processes
  - Individual execution threads
- Preinstrumented executable files

  Preinstrumenting allows you to write profile selection settings (instrumentation) to the current executable file or to a copy of the current executable file. You can run the preinstrumented executable file outside the control of CXperf. The profiling data is collected in a performance data file (PDF) for later analysis.
- Graphical analysis for performance data
- Profiling MPI and PVM message passing applications

Use CXperf to discover which routines or loops slow down a program's execution. In some cases, simple modification of source code, such as inserting compiler directives, results in significant performance improvements. Profiling versions of a program that have been compiled at different optimization levels provides insight into the types of optimizations that work best for a given situation.

Using CXperf

To invoke CXperf, type `cxperf` at your UNIX prompt with or without specifying a file name. The file name can be an executable file or a Performance Data File (PDF). Refer to “Using Performance Data Files (PDFs)” on page 83 for details about PDFs. The performance analysis process consists of four steps:

- **Compilation**—Although it does not aid in compiling a program for profiling, CXperf provides instructions for compiling because correct compilation is the important first step in the profiling process. Refer to “Compiling” on page 22 for compiling instructions.
- **Instrumentation**—Selecting regions to profile and metrics to collect. Refer to Chapter 4, “Choosing Data,” for details.
- **Execution**—Running an instrumented executable file, either within CXperf or outside. Refer to Chapter 2, “Getting started,” for details about starting CXperf in line mode or GUI mode.
- **Analysis**—Extracting and understanding data gathered in a PDF during execution. Refer to Chapter 6, “Analyzing,” for more details.

Profiling is an iterative process; you profile a program, make changes to the source code based on the results, and profile again.

When you start CXperf with no filename, the Compilation Page appears. The Compilation Page provides compile information for profiling and allows you to browse a file list to choose an executable file.

When you start CXperf with an executable file, the Instrumentation Page appears.

When you start CXperf with a PDF, the Analysis Page appears.

All pages have a file menu and a help menu. Use the file menu to set preferences, open new files, and exit CXperf. Use the help menu to access online help.

CXperf interfaces

You can run CXperf in X/Motif Graphical User Interface (GUI) mode, character oriented tty interface (line) mode, and batch mode.

You can use more than one mode for a single profiling task. For example, you can run an application in line mode or batch mode to collect profiling data, then use the GUI to view graphical analysis of the data.

Text reports are available in all modes.

GUI mode

To invoke CXperf in GUI mode, type `cxperf` at your UNIX prompt. Navigate through the profiling process using the **Previous** and **Next** buttons at the bottom of each page. You may be automatically moved between pages as the profiling process progresses; you may be prevented from moving to a page under certain conditions. By guiding you through the process in this way, profiling with CXperf becomes a straightforward and intuitive process.

CXperf provides graphical analysis of performance data through the GUI. The GUI provides:

- Mouse-driven selection of region types to profile and metrics to collect.
- Intuitive, step-by-step guidance through the compile, instrument, execute, and analyze steps.
- Summary Profile (2D) and Parallel Profile (3D) graphs to analyze performance data.
- Call Graphs with point-and-click navigation.
- Source code correlation when you click on a bar of the Summary or Parallel Profile graph or on a node in the Call Graph.
- Source code display facility with source code annotations indicating regions profiled.
- Text performance report functionality on the Analysis Page.
- PDF analysis for files created on a different architecture.

- Multiple Analysis Page comparisons which allow you to:
  - Compare and contrast profiling data for different metrics.
  - View data from multiple PDFs simultaneously.

Line mode

Line mode is a character based, command line interface for CXperf. To use line mode, specify the `-nw` option when you invoke CXperf from the UNIX prompt. When you start CXperf in line mode with the name of a PDF, use the `set pdf` and `analyze` commands to access performance data for multiple PDFs, including PDFs created on different architectures.

Line mode presents performance data in Text reports only. However, after you collect profiling data in line mode, the resulting PDF can be analyzed in GUI mode.

Batch mode

Batch mode allows you to run CXperf by incorporating it in a script or text file. You make use of CXperf's tty commands to profile applications in batch mode.

To use batch mode, provide a command file using the `-x` option on the command line, or invoke CXperf from a shell script, or both—provide the command file within a shell script. You can redirect input, output, and standard error to and from files.

Refer to “Batch mode” on page 85 for more information.

Graphical analysis

CXperf provides Summary Profile, Parallel Profile, and Call Graph analysis of profiled data. Each graphic analysis page has the following capabilities:

- **Source code correlation**—Click on any bar in the Summary or Parallel Profile, or any node in the Call Graph, to display the source code associated with the routines being graphed.
- **Zooming options**—Use the Zoom feature with the Summary and Parallel Profiles when you have a large number of data items to graph and you want to focus on a subset. Use the Collapse and Expand feature to vary the number of routines displayed on the Call Graph.
- **Tear-off analysis**—Use the Tear-off analysis feature to open multiple graphs to compare and contrast profiling data simultaneously.
- **Graph Configuration**—Configure your graphs during analysis using the Region and Metric sections at the top of the Analysis Page.

Refer to Chapter 6, “Analyzing,” for more information about graphical analysis.

Summary Profile and Parallel Profile

Two-dimensional and three-dimensional graphical analysis—Summary Profile and Parallel Profile, respectively—are available only in GUI mode.

The Summary Profile graphs the data per routine, while the Parallel Profile graphs the data per thread and per routine. Graphical analysis is interactive. You can select the specific region types and metrics to graph for each profile.

Summary Profiles and Parallel Profiles provide the following features in addition to the general features listed above:

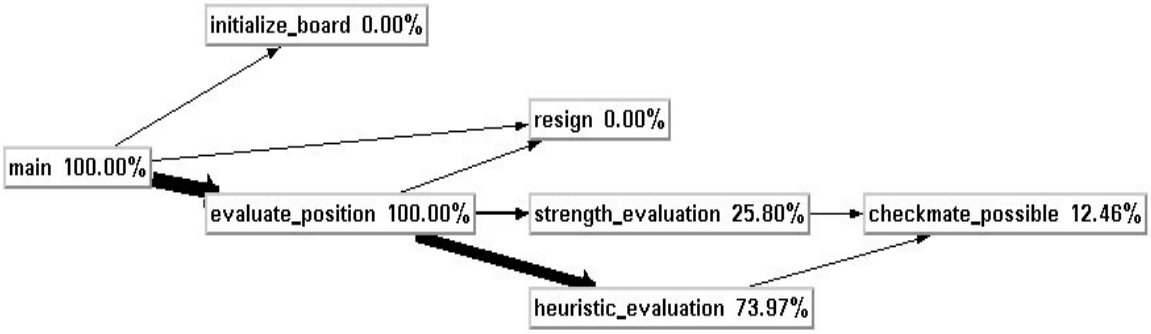
- **Saving profiles for printing or export**—Save graphs in PostScript or XWD formats for printing, or in ASCII format for export to other graphics packages.

- **Parallel Profile graph rotation**—Rotate the graph by placing the mouse pointer over the graph and moving it using the middle mouse button. To restrict the rotation to a single axis, press the **x**, **y**, or **z** key while you move the mouse to rotate the graph.

Call Graph

The Call Graph is a graphical representation of the relationships between routines in a program. A typical Call Graph is shown in Figure 1.

Figure 1 Call Graph



Each node of the graph represents a routine in the program. The nodes are labeled with the routine name and the specific metric value for that routine. The metric value specified for each routine represents a percentage of the total value for that metric contributed by the particular routine.

Arrows between the nodes the call graph indicate caller and called routines. The arrow points from the caller routine to the one called. The critical path through the program is shown by thicker arrows along that path.

Performance Reports

Text performance reports are available in both GUI and line mode. Metrics available in performance reports vary according to machine architecture, region types selected during instrumentation, and the options used when you compiled your program.

In GUI mode, CXperf displays Summary and Parallel Reports. Summary Reports display profiling data for the whole application. Parallel Reports have finer granularity, displaying data for each process and for all threads in each process.

Text reports are similar in GUI and line mode, providing Performance Analysis sections for:

- Routines.
- Loops (All)—Including compiler-generated parallel loops, for modules compiled with HP compilers at optimization levels `+O2` and `+O3`.
- Loops (Parallel only)—Parallel loops generated by HP compilers at optimization level `+O3 +Oparallel`.

Metrics

Collecting and comparing different metrics helps identify performance bottlenecks such as:

- Routines and loops that consume the most Wall Clock and CPU time
- Regions of code that spend a significant amount of their CPU time waiting for memory
- Loops that generate excessive cache misses
- Uneven distribution of work across threads in parallel regions
- Lack of effective parallelism in a loop or a routine
- Memory bank contention or cache thrashing among threads in parallel regions

The type and number of metrics available differ according to machine architecture. In addition to the defaults—Wall Clock time, CPU time, and Execution counts—a number of metric groupings are available. In CXperf, the available metrics are grouped based on functionality. The groups are:

Timer events	Wall clock time, CPU time, execution counts
Process events	Migrations, context witches, (voluntary and involuntary), page faults
Memory events	Data TLB misses, instruction TLB misses, cache misses, latency, instruction counts
Data Cache Utilization	Cache misses, latency, instruction counts
Data and Instruction TLB misses	Data TLB misses, instruction TLB misses, Instruction counts

See “Introducing metrics” on page 42 for more detailed information about available metrics.

2 Getting started

This chapter provides information to allow you to get started quickly using CXperf. You work through a profiling session and use CXperf's standard features in GUI mode and line mode for each step of the process. Topics covered include:

- Overview of a profiling session
- Profiling a program in GUI mode
  - Compiling
  - Instrumenting
  - Executing
  - Analyzing
- Profiling a program in line mode
  - Compiling
  - Instrumenting
  - Executing
  - Analyzing
  - Editing the command line

Overview of a profiling session

To profile a program using CXperf, follow four fundamental steps:

Step 1. Compile.

Compile your program with the +pa or +pal CXperf option, at optimization levels +O2, +O3, or +Oparallel.

Step 2. Instrument.

Select the metrics you want to collect and the source code regions—routines, loops, or parallel loops—at which you want to collect them.

Step 3. Execute.

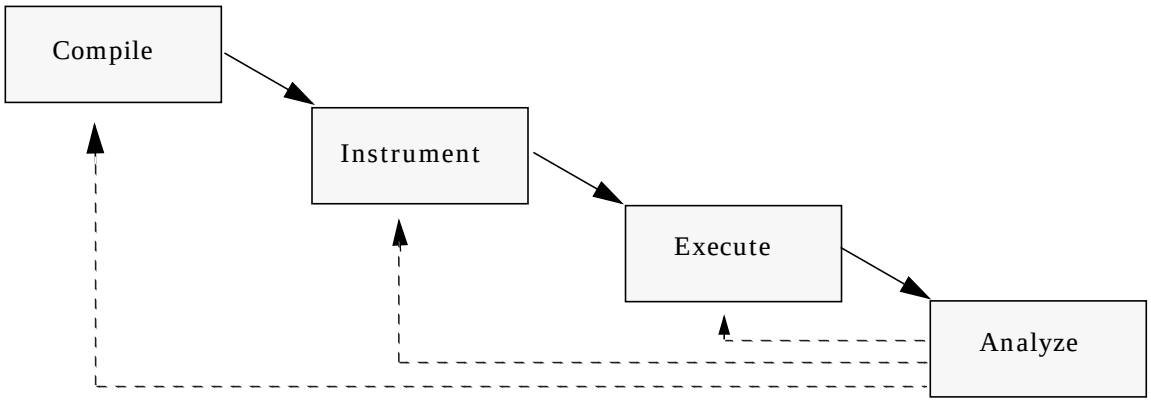
Run your instrumented program under the control of CXperf, or by exiting CXperf and running the executable file. CXperf creates a Performance Data File (PDF) containing the profiling data.

Step 4. Analyze.

Analyze the contents of the PDF.

Figure 2 outlines the four-step procedure.

Figure 2 Profiling using CXperf



You can profile versions of a program that have been compiled at different optimization levels to gain insight into the types of optimizations that work best for a given situation.

However, as indicated in Figure 2, you do not need to recompile your program to select a different set of metrics to collect, or a different set of region types at which to collect them. You can return to the Instrumentation step during a profiling session and select different options.

The following sections take you step-by-step through a profiling session. You learn the basics, in GUI and line mode, for the four profiling steps:

- Compiling
- Instrumenting
- Executing
- Analyzing

Profiling a program in GUI mode

The following sections present a minimalist procedure for profiling a program in GUI mode.

Compiling

CXperf does not actually aid you in compiling, but, in GUI mode, provides instructions for compiling programs using HP compilers. Refer to Table 2 for compiling instructions.

The compiler you use to build programs for profiling with CXperf depends upon the programming language you used. CXperf is an interactive runtime performance analysis tool for programs compiled with the HP Parallel compilers shown in Table 1.

Table 1 **Compilers**

Language	HP compiler
Fortran 90	/opt/fortran90/bin/f90
Fortran 77 (Exemplar 32-bit)	/opt/fortran/bin/f77
ANSI C	/opt/ansic/bin/c89
ANSI C++	/opt/aCC/bin/aCC

Step 1. Start CXperf by typing `cxperf` with no command line options at your UNIX prompt.

```
% cxperf
```

The Compilation Page displays instructions for compilation.

Step 2. Read the compile instructions on the Compilation Page.

Decide which compile preference you need. For this example, compile and link in a single step.

You must go to a UNIX prompt to compile your program.

Step 3. Compile and link in a single step to analyze routines and loops.

For example, using the ANSI C compiler enter:

```
% /opt/ansi/bin/c89 +pal +O3 myprogram
```

When compilation completes, you have an executable file called a.out.

Refer to Chapter 3, “Preparing programs to profile,” for details about compiling source, object, and library files for CXperf.

Instrumenting

To instrument your executable file, a.out, you first need to invoke CXperf.

To start CXperf and instrument a.out in GUI mode, follow this procedure:

Step 1. Set your DISPLAY environment variable. For example, using C shell syntax enter:

```
% setenv DISPLAY display_name:0.0
```

where *display_name* is your terminal name.

If your display variable is not set, CXperf displays a message and starts in line mode.

Step 2. Invoke CXperf with the name of your executable file.

```
% /opt/cxperf/bin/cxperf a.out &
```

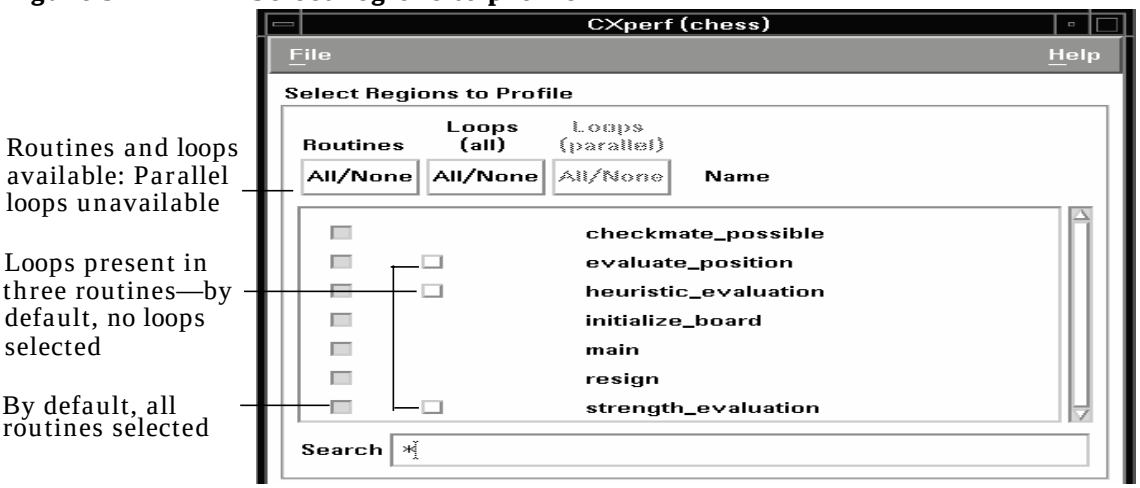
CXperf opens on the Instrumentation Page.

Step 3. Select regions to profile.

Because you compiled a.out with the `+pal` option, routines and loops are available for instrumentation. By default, all routines and no loops are selected for profiling.

Figure 3 displays the top section of the Instrumentation Page. Use this section to select routines and loops to profile.

Figure 3 Select regions to profile



The program whose routines are displayed in Figure 3 has seven routines, three of which contain loops. You can select all loops, or loops in specific routines, by selecting the buttons that are adjacent to `evaluate_position`, `heuristic_evaluation`, and `strength_evaluation`. You can use the **All/None** button under **Loops(all)** to select or deselect all loops in the program.

If you have a large program, do not select all routines and all loops to profile in a single session, because the more region types and metrics you select to profile, the slower your code executes. Refer to “Profiling strategy” on page 74 for a discussion of profiling intrusion.

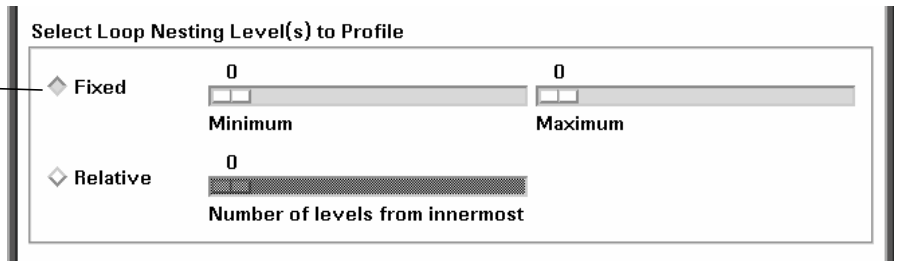
You need not recompile a program to change the selections of regions to profile. Return to the Instrumentation Page and change your selections.

Step 4. Select Loop Nesting Levels.

Use the second section on the Instrumentation Page as shown in Figure 4 to select the loop nesting level.

Figure 4 Select loop nesting level to profile

By default, selects all loops with a nesting level of 0



For an initial profiling session, use the default setting which specifies a fixed loop nesting level range with a minimum of 0 and a maximum of 0. All loops with a nesting level of 0 after optimization—outermost loops—are selected for profiling.

Selecting only outermost loops minimizes profiling intrusion and is useful for an initial profiling session. Refer to “Selecting loop nesting levels” on page 54 for details about other loop nesting options.

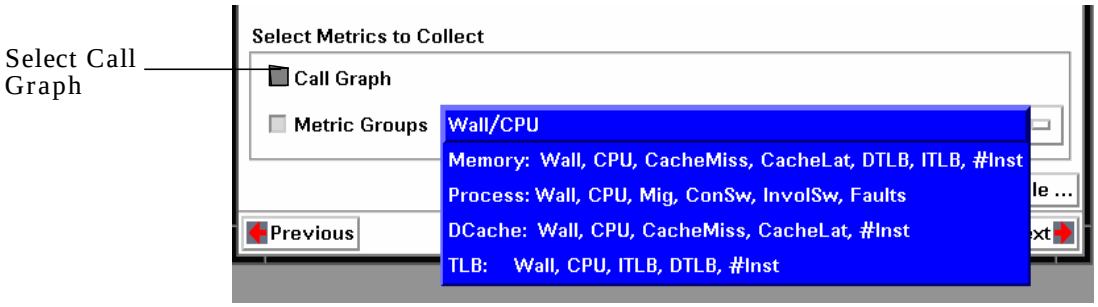
Step 5. Select metrics to collect.

The type and number of metrics available differ according to machine architecture. Refer to “Introducing metrics” on page 42 for details.

Figure 5 displays the lower section on the Instrumentation Page. Memory events, Process events, Data Cache Utilization (DCache), and Data and Instruction TLB misses (TLB) are all available in Figure 5, indicating this program is instrumented on an HP V-Class server.

Wall Clock time and CPU time are the defaults and are always collected. Select one other metric group to collect.

Figure 5 Select Metrics and Call Graph


Select Call Graph
Select Metrics to Collect
Call Graph
Metric Groups
Wall/CPU
Memory: Wall, CPU, CacheMiss, CacheLat, DTLB, ITLB, #Inst
Process: Wall, CPU, Mig, ConSw, InvolSw, Faults
DCache: Wall, CPU, CacheMiss, CacheLat, #Inst
TLB: Wall, CPU, ITLB, DTLB, #Inst
Previous

Call Graph is deselected by default. Select Call Graph if you wish to analyze Call Graphs at the Analysis step.

Step 6 and “Executing” on page 19 describes how to run the program under CXperf control.

Alternatively, you can select **Preinstrument Executable** at the bottom of the Instrumentation Page to write the instrumentation selections you just made to the executable file, or to a copy. You can then exit CXperf and run the executable file outside CXperf to generate a Performance Data File (PDF). Refer to “Preinstrumenting” on page 66 for details.

Step 6. Click **Next** to go to the Execution Page.

CXperf displays the Execution Page.

Executing

When you finish instrumenting your program on the Instrumentation Page and select the **Next** button, CXperf opens the Execution Page. Figure 6 displays the Execution Page.

Figure 6 Execution Page

To execute your program, follow this procedure:

Step 1. Press **Start**.

The Process State changes from **Not started** to **Running**. A status window with program information appears while the program is running.

Step 2. Wait for the program to complete running.

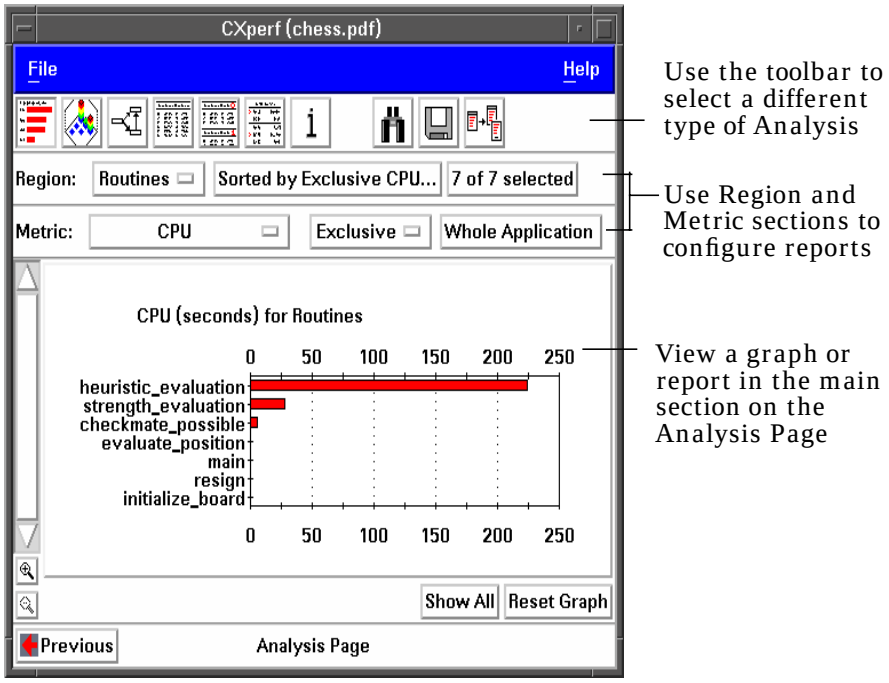
The **Pause**, **Continue**, and **Abort** buttons are available during the program run. For the most accurate results, do not pause your program during profiling.

When the program completes, CXperf exits the Execution Page and opens the Analysis Page.

Analyzing

When your program finishes its run, the Analysis Page appears, displaying the performance data in a Summary Profile. Figure 7 displays the Analysis Page.

Figure 7 Analysis Page



The Analysis Page toolbar menu and pulldown menus allow you to select different types of data analysis and other report configuration options.

When you choose a mode of analysis, the appropriate graph or text report appears on the Analysis Page.

The following graphical and text reports are available to analyze your profiling data:

- Summary Profile
- Parallel Profile
- Call Graph, if you selected Call Graph during Instrumentation

- Summary and Parallel Reports
- Call Graph Report, if you selected Call Graph during Instrumentation

Refer to “Analysis Page” on page 90 and “Configuration options” on page 95 for details about the functionality available to help you analyze your data.

Profiling a program in line mode

The following sections present a minimalist procedure for profiling a program in line mode.

Compiling

Although CXperf does not actually aid you in compiling, it provides instructions for compiling programs using HP compilers. To see the Compilation Page with the instructions you must start CXperf in GUI mode. Enter `cxperf` at your UNIX prompt:

```
% cxperf
```

The Compilation Page appears, displaying instructions for compilation.

Table 2 lists instructions for you to compile your program. Refer to Table 1 on page 14 for a list of supported HP parallel compilers.

Step 1. Read the compile instructions in Table 2.

Step 2. Decide which compile preference you need. For this example, compile and link in a single step.

Step 3. Compile and link in a single step to analyze routines and loops.

For example, using the ANSI C compiler enter:

```
% /opt/ansi/bin/c89 +pal +O3 myprogram
```

When compilation completes, you have an executable file, a.out, ready for profiling with CXperf. Your profiling options are determined by the compilation you just performed:

- `+pal` compiles the program for routine and loop level profiling.
- `+O3` specifies a compiler optimization level that supports profiling routines and loops.

Table 2 describes compiling command syntax to use for programs compiled with HP ANSI C (c89), ANSI C++ (aCC), Fortran 90 (f90), and Fortran 77 (f77) compilers.

Table 2 Compile instructions

Function	Command syntax
Compiling in a single step to analyze routines	*compiler* `+pa` `-o` *executable source_files*
Compiling in a single step to analyze routines and loops	*compiler* `+pal` {`+O2`|`+O3`} `-o` *executable source_files*
Compiling and linking to analyze routines	*compiler* `-c +pa` *source_file* `-o` *object_file*
	compiler `+pa -o` *executable object_files*
Compiling and linking to analyze routines and loops	*compiler* `-c +pal` {`+O2`|`+O3`} *source_file* `-o` *object_file*
	compiler `+pal -o` *executable object_files*
Linking to analyze routines	`cxoi` *object_files libraries* `-o` *executable*
	compiler `+pa -o` *executable object_files libraries*

Refer to Chapter 3, “Preparing programs to profile,” for more details about compiling source, object, and library files for CXperf.

Instrumenting

To instrument your executable file, first invoke CXperf.

To start CXperf and instrument a.out in line mode, follow this procedure:

Step 1. Invoke CXperf with the name of your executable file and the `-nw` (no windows) option.

```
% /opt/cxperf/bin/cxperf -nw a.out

                Convex Performance Analyzer

Type 'help' for help.

Reading executable a.out...

Selecting profile a.out.pdf...

(CXperf)
```

As shown in the output for the command above, CXperf displays the name of the executable file to be profiled and the name of the PDF that the performance data is written to. By default, the PDF is named *executable*.pdf.

NOTE Use the CXPERF environment variable to specify command line options for starting CXperf. For example,

```
% setenv CXPERF '-nw -pid -w'
```

forces CXperf to start in line mode (-nw). The -pid option specifies that CXperf add the process ID number of the process you are profiling to the name of the PDF it creates. The -w option suppresses warning messages issued by CXperf.

Step 2. Select regions to profile with a form of the `select` command.

The select command syntax is as follows:

```
select [ routine | loop ] all
```

where

`routine`	Selects routines to profile.
`loop`	Selects loops to profile.
`all`	Instructs CXperf to select all routines in your program for profiling.

In this example enter:

```
(CXperf) select all
```

Because a.out is compiled with the +pal option, for this example, routines and loops are available for instrumentation.

The all parameter instructs CXperf to select all routines in your program for profiling.

Both routines and loops are selected for profiling because you did not use the [routine |loop] parameter to specify only one region type.

In line mode, if you do not use select to select one or more source code regions for profiling, CXperf does not collect any metrics. Refer to "Selecting routines and loops" on page 59 for details about using select.

If you have a large program, do not select all routines and all loops to profile in a single session, because the more region types and metrics you select, the slower your code executes. Refer to "Profiling strategy" on page 74 for a discussion of profiling intrusion.

Step 3. Select metrics to collect with the collect and set events commands.

```
(CXperf) collect cpu wall_clock call_graph events
```

```
(CXperf) set events process
```

collect instructs CXperf to collect

- CPU time
- Wall Clock time
- Call Graph
- events (Specifies collecting one metric set available on the current architecture. Memory, Process, Data Cache Utilization, and Data and Instruction TLB events are possibilities.)

Use set events immediately after collect events to specify which events to collect. For this example, set events process instructs CXperf to collect Process events.

The type and number of metrics available differ according to machine architecture. Refer to "Introducing metrics" on page 42 for details. For example, if you run this program on an HP V-Class server, you can use any one of the set events command in Table 3.

The `set events` command options available when you run your program on an HP V-Class server are shown in Table 3.

Table 3 `set events` options

Command	Specifies
`set events memory`	Memory events*
`set events process`	Process events**
`set events tlb_misses`	Data and Instruction TLB misses*
`set events data_cache`	DataCache Utilization
* These metrics can only be specified and collected on HP V-Class servers K-Class and D-Class servers	
**These metrics can only be specified and collected on HP K-Class and D-Class servers	

“Executing” on page 27 describes the next step—how to run the program under CXperf control.

Alternatively, you can write the instrumentation selections you just made to the executable file, using the `save executable` command. You can then exit CXperf and run the executable file to generate a Performance Data File (PDF). Refer to “Preinstrumenting in line mode” on page 70 for further details.

Executing

This section describes how to execute your program under CXperf control in line mode.

Step 1. Run your program using the `run` command.

```
(CXperf) run
```

The run command syntax is:

`run` [*argument* ...] [*i/o_redirection*]

where

argument	Specifies any number of command line arguments to the program you are profiling. Separate multiple arguments with spaces.
io_redirection	Redirects the standard input, output, or error from or to the specified file when you use one of the redirection operators (<, >, >>, >&, >>&).

Step 2. Wait for the program to complete running.

Your program runs to completion unless you press **CTRL-C** to pause it. For the most accurate results, do not pause your program during profiling.

When you pause a program, use `continue` to resume execution, or `stop` to terminate the program. Refer to the *CXperf Command Reference* for details about CXperf line mode commands.

Analyzing

Use `analyze` to view performance reports after your program finishes. For example

`(CXperf)` **`analyze`**

creates a performance report from the PDF that CXperf created. By default, the PDF is named a.out.pdf.

When you use the `analyze` command without specifying any parameters, CXperf generates and displays all available performance reports.

CXperf displays reports using the pager specified with your PAGER environment variable. If your PAGER environment variable is not set, CXperf uses the `more` command to page the output.

Refer to “Line Mode Report” on page 131 for details about the output of `analyze`.

Editing the command line

CXperf's line mode provides command line editing functions similar to those available in `tcsh`. Enter **ESC-?** on the CXperf command line to display available editing functions. Table 4 lists the command line editing functions available for CXperf.

Table 4 lists editing functions for the CXperf command line.

Table 4 Editing the command line

Function	Key sequence
Backward character	**CTRL-b**
Backward word	**ESC-b**
Beginning of line	**CTRL-a**
Capitalize forward word	**ESC-c**
Delete backward character	**CTRL-h**
Delete backward character	**DEL**
Delete backward word	**ESC-h**
Delete forward character	**CTRL-d**
Delete forward word	**ESC-d**
Display key bindings	**ESC-?**
End of line	**CTRL-e**
Erase line	**CTRL-g**
Erase screen	**ESC-g**
Execute current command	**RETURN**
Execute a shell command	**!<command>**
Forward character	**CTRL-f**
Forward word	**ESC-f**
Kill to end of line	**CTRL-k**
Lower case word	**ESC-l**
Next command	**CRTL-n**
Previous command	**CTRL-p**
Transpose characters	**CTRL-t**
Transpose words	**ESC-t**

3 Preparing programs to profile

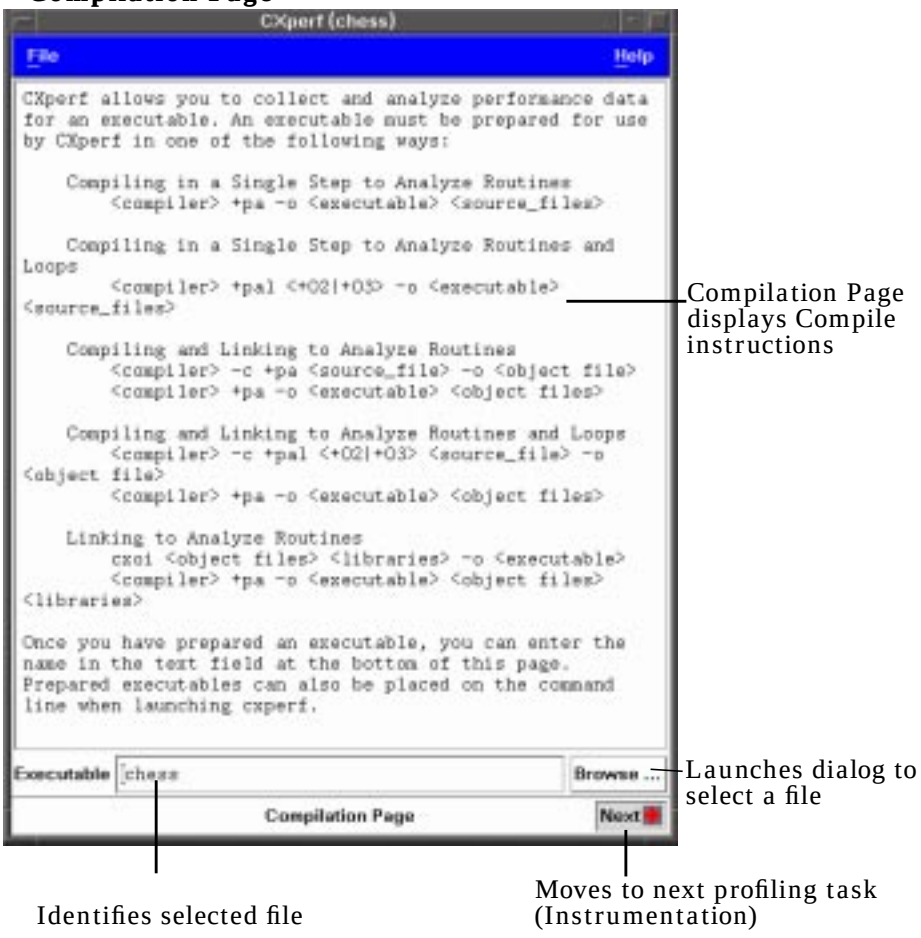
This chapter describes the methods you use to prepare a program for profiling. First, you are introduced to compiling options for preparing standard binary files. You become familiar with CXoi, a utility that prepares object or archive library files for profiling. Topics covered include:

- Compiling
  - `+pa` and `+pal` options
  - Syntax
  - Compiling and linking in one step
  - Compiling and linking separately
- Using CXoi to instrument object files and archive libraries
  - Syntax
  - Preparing for profiling
  - CXoi limitations

Compiling

The first step in the performance analysis process is compilation. CXperf does not actually aid you in compiling, but provides instructions—in GUI mode—for compiling programs using HP Fortran 90, ANSI C++, ANSI C, and HP Parallel 32-bit Fortran 77 compilers. To see compiling instructions, start CXperf by typing `cxperf` with no command line options at the command prompt. The Compilation Page containing compile instructions, as shown in Figure 8, appears.

Figure 8 Compilation Page



Use the **Browse** button on the Compilation Page to browse a list of files. The **Browse** button launches a dialog as shown in Figure 9. Select an executable file in the dialog.

Figure 9 **Browse: Select a file**

Browse the directories and files and select the file you want to profile.

+pa and +pal options

To compile and link an application for profiling with CXperf, specify the +pa or +pal compiler option. The +pa option instructs the compiler to instrument routines for profiling. The +pal option instructs the compiler to instrument routines and loops. The compiler adds instructions to the executable file, enabling CXperf to gather performance data during execution of the program. Inserting instructions into the executable file is known as instrumenting the file. Specify the +pa or +pal option when linking to ensure that timing and data collection routines —namely, cxperfmon.o—link into the executable.

The source code regions you select for profiling depend on the compiler optimization level you specify. Optimization options are:

- +O0 and +O1—Select only routines for profiling. +O0 is the default optimization level for HP compilers.
- +O2 and +O3—Select routines and loops for profiling.

The following compiler options are incompatible with +pa or +pal:

- -p and -G
- +O4, +Oall, and +Oprocelim
- -s

Syntax

To compile and link an application for profiling, use the following syntax:

compiler { `+pa | +pal` } [*optimization_options*] *files*

where

compiler	Specifies one of the HP compilers: /opt/fortran90/bin/f90—Fortran 90 /opt/fortran/bin/f77—Fortran 77 /opt/aCC/bin/aCC—ANSI C++ /opt/ansic/bin/c89—ANSI C
optimization_options	Specifies the compiler optimization level. The region types that may be profiled depend on the optimization level: `+O0` and `+O1`—Routines can be profiled. `+O2` and `+O3`—Routines and loops can be profiled. `+Onoinline`—Suppresses inlining. `+O4`,`+Oall`, and `+Oprocelim`—Not supported for use with `+pa` and `+pal`.
`+pa`	Compiles the application for routine-level profiling.
`+pal`	Compiles the application for routine- and loop-level profiling.
files	Specifies the name of one or more source files, object files, or libraries.

Refer to the *Parallel Programming Guide for HP-UX Systems* for more details about compiler optimization levels.

NOTE To profile Fortran 77 programs, you must use the HP Parallel 32-bit Fortran 77 compiler. CXperf version 6.0 does not support the standard HP Fortran 77 compiler.

Compiling and linking in one step

If you compile your source file into an executable file with a single call to the compiler, you compile and link in the same step. When you compile and link in one step, object files are not saved, and the executable file is ready to be used by CXperf.

The following example compiles and links the source file in a single step:

```
% /opt/fortran90/bin/f90 +pal +O3 +Onoinline main.f
```

In the example above:

- The source file main.f compiles at optimization level `+O3` with the `+pal` compiler option to produce the executable file a.out.

  Routines and loops are instrumented for profiling with CXperf because the `+pal` option is specified and the `+O3` optimization level is used.

- The `+Onoinline` option suppresses inlining.

  At optimization level `+O3` the HP parallel compilers can inline routines called within the same source file.

  Inlining substitutes selected function calls with copies of the function's object code. Inlining may result in larger executable files and greater compilation time.

  If you compile your program with the `+O3` option (not adding the `+Onoinline` option) and find that only a subset of your instrumented routines are available during analysis, it is likely that those routines that are not available are inlined during your program run.

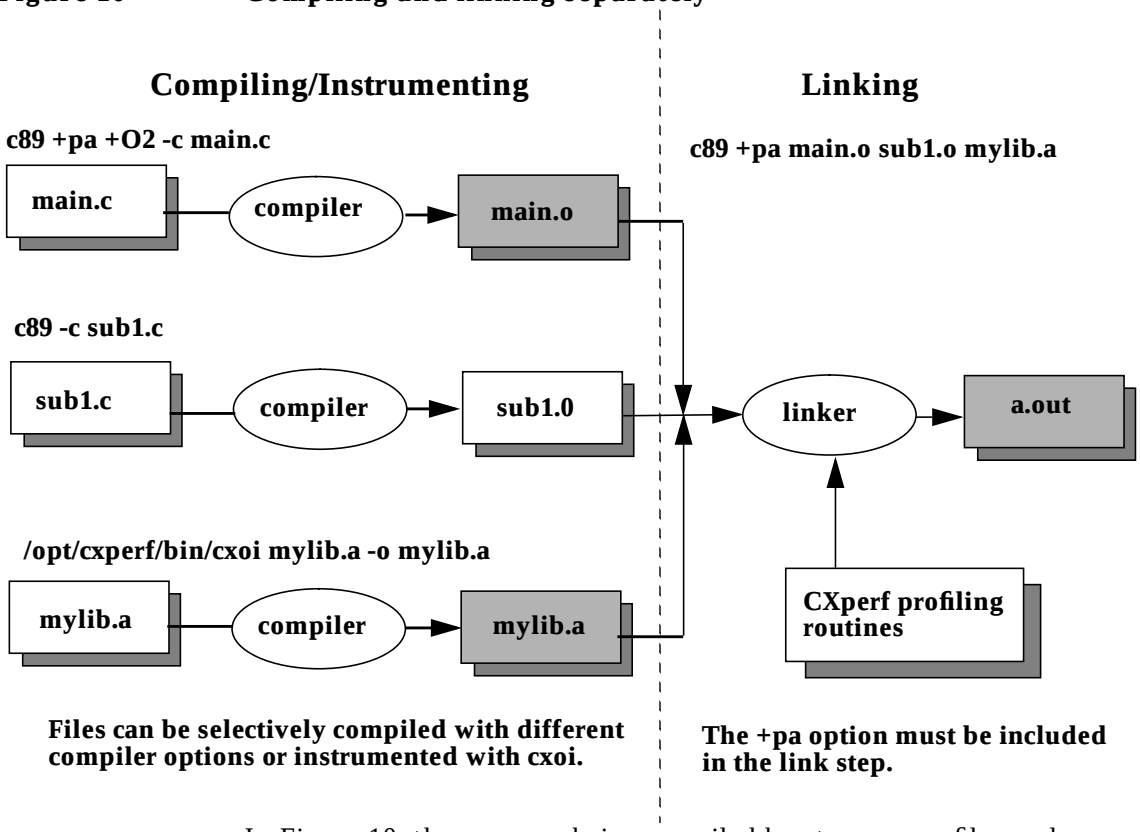
Compiling and linking separately

Typically, when there are a large number of source files for a program they are compiled separately. Each source file is compiled into an object file using the `-c` compiler option (to suppress linking) and then linked together into an executable file.

When compiling for CXperf, you can compile each source file with the same or different options. However, you must use the `+pa` or `+pal` option when linking.

Figure 10 demonstrates the separate steps of compiling and linking.

Figure 10 Compiling and linking separately



In Figure 10, the program being compiled has two source files and an archive library. In the compiling and instrumenting phase:

- The source file main.c is compiled into an object file at optimization level +O2.

  The +pa option instruments the file for routine level profiling with CXperf. The -c option suppresses linking.

- The source file sub1.c is compiled into an object file without adding any instrumentation for CXperf.

- The archive library mylib.a is instrumented for profiling with CXoi, the object and archive library file instrumentor.

  The -o option specifies the name of the instrumented file.

In the linking phase shown in Figure 10 there is a second call to the compiler, as follows:

```
% c89 +pa main.o sub1.o mylib.a
```

This invokes the linker, which in turn combines instrumented object files and archive library files into an executable file. The linker also links the CXperf timing routines (cxperfmon.o) into the executable file. You cannot profile using CXperf unless these routines are linked into the executable file.

Using CXoi to instrument object files and archive libraries

CXoi is a separate utility shipped with CXperf. It is an object file and archive library instrumentor you use to instrument files produced by any PA-RISC targeting compiler. Only routine level profiling is possible with CXoi.

Syntax

To instrument an object file or an archive library file for profiling, use the following syntax:

`cxoi` { *lib.a* | *file.o* } [`-o` *output_file*] [`-tx`, *name*]

where

lib.a	Specifies an archive library file.
file.o	Specifies an object file. You can specify only one per invocation of Cxoi.
`-o` *output_file*	Specifies the file to write the instrumented *file.o* or *lib.o* to. If you do not specify the `-o` option, CXoi names the instrumented file *file*.cxoi.o or *lib*.cxoi.a.
`-tx`, *name*	Specifies the path *name* for a linker, an assembler, or both. Use when you want to use a different linker or assembler than the default. The `x` identifier takes one or more of the following values: `a`—Assembler (standard suffix is `as`). `l`—Linker (standard suffix is `ld`). If `x` is a single identifier, *name* represents the full path name of the linker or assembler. If `x` is a set of identifiers, *name* represents the path to which the standard suffixes are concatenated to construct the full path names for the assembler and linker.

Preparing for profiling

Instrumenting with CXoi

Use the CXoi utility to insert instrumentation instructions into object files or archive library files compiled with PA-RISC compilers. Only routine level profiling is supported by CXoi.

The examples below demonstrate using CXoi to insert instrumentation instructions for collecting routine-level performance information into an object file (file.o) and an archive library (libx.a), respectively.

```
% /opt/cxperf/bin/cxoi file.o
```

```
% /opt/cxperf/bin/cxoi libc.a
```

By default, CXoi names the instrumented object or library file *file*.cxoi.o or *libc*.cxoi.a. To specify a different name for the instrumented file, use the `-o` option:

```
% /opt/cxperf/bin/cxoi libc.a -o mylibc.a
```

In the example above the `-o` option creates a new archive library file, mylibc.a. The new file is a copy of libc.a but additionally contains CXperf instrumentation instructions for routine entry points. The original file, libc.a, is not modified.

To modify the original object file or library file in place, you must have write permissions to the file and its parent directory. Specify the original filename with the `-o` option.The original library file gets overwritten with a version instrumented for profiling with CXperf.

You cannot specify multiple object files or libraries with CXoi. For example, the following commands do not work:

```
% /opt/cxperf/bin/cxoi *.o
```

```
% /opt/cxperf/bin/cxoi obja.o objb.o
```

Linking the instrumented files

After using CXoi to instrument the object or archive library files, link the instrumented files into an executable file using the `+pa` option supported by HP compilers. The examples below demonstrate the syntax:

```
% /opt/fortran90/bin/f90 +pa file.cxoi.o
```

```
% /opt/fortran90/bin/f90 +pa file.o libx.cxoi.a
```

If CXoi encounters an object file already instrumented for CXperf, it ignores the file, displays a warning message, and exits. If you are instrumenting an archive library and CXoi enters an object file that is already instrumented, CXoi ignores the object file and continues instrumenting the other object files in the archive.

CXoi limitations

Although CXoi is a useful utility to instrument object files and archive libraries, it has the following limitations:

- CXoi cannot be used to instrument shared libraries.
- CXoi supports routine-level but not loop-level profiling.
- CXoi requires space in /usr/tmp—or in the directory specified by the environment variable TMPDIR—totaling at most three times the size of the file being instrumented.

  If /usr/tmp does not have the required amount of space, set your TMPDIR variable to a different directory with sufficient space.
- Routines whose names begin with one or more leading underscores (_), millicode, and routines declared static in C or C++ are never exposed for profiling.
- CXperf does not support source code correlation for any routine exposed for profiling using CXoi.
- Object files and archive libraries instrumented for profiling with CXoi do not contain source file line number information.

  Source code correlation for routines within these modules always refers to line 1 of the source file that contains the routine. CXperf source code annotations are not displayed in the Source Code window or in source file listings for object files and libraries instrumented with CXoi.
- CXperf may display the following error message:

  `ERROR D5: Cannot find symbolic support in current executable.`

  Ignore this message. Performance analysis is not affected.

4 Choosing Data

In this chapter you learn the methods for selecting region types and metrics to profile, whether you use CXperf in GUI or line mode. This chapter also describes the types of metrics available. You learn how to write profile selection settings to a program, which is called instrumenting. You also learn how and why to preinstrument a program. Topics covered include:

- Introducing metrics
  - Metrics available on all architectures
  - Architecture-dependent metrics
  - Using event metrics
- Instrumenting
  - Instrumenting in GUI mode
  - Instrumenting in line mode
- Preinstrumenting
  - Setting the environment
  - Preinstrumenting in GUI mode
  - Preinstrumenting in line mode

Introducing metrics

You can specify the types of performance metrics to collect for each of the source code regions you profile. Collecting and comparing different metrics helps identify performance bottlenecks, such as

- Routines and loops that consume the most Wall Clock and CPU time
- Regions of code that spend a significant amount of their CPU time waiting for memory
- Loops that generate excessive Cache misses
- Uneven distribution of work across threads in parallel regions
- Lack of effective parallelism in a loop or a routine
- Memory bank contention or cache thrashing among threads in parallel regions

The type and number of metrics available differ according to machine architecture. In addition to the Timer metrics, comprising Wall Clock time and CPU time, a number of other metric groupings are available. The available metric groups are based upon functionality. Refer to “Architecture-dependent metrics” on page 44 for more details about the groupings.

The following sections describe the different metrics available when profiling with CXperf.

Metrics available on all architectures

Timer metrics are the default metric set, and are available on all architectures. The following list describes the metrics that are collected by CXperf as part of the Timer metric set:

CPU Time
: Time the processors work on the process, not including time waiting for I/O or running other programs. If a process can run multiple processors, the CPU time may be greater than the Wall Clock time.

Wall Clock
: Time to solution, including process idle time.

Execution Counts
: Number of times a routine executes, or for loops, the number of loop invocations.

Call Graph
: Wall clock time and CPU time (inclusive and exclusive of child processes), Execution counts, and metrics for each profiled routine, its parents, and its children.

CPU/Wall Clock
: Ratio of CPU to Wall Clock time. This is a derived metric, computed during analysis. The interpretation of this ratio depends on the region type profiled:

  For serial regions, if the CPU/Wall Clock ratio is high (approaches 1.0), the region is compute-bound.

  For parallel regions, the ratio indicates the *concurrency factor*, or the increased speed achieved through parallelization. Values approaching *n*, where *n* is the number of processors the program runs on, indicate good parallel concurrency.

  For both parallel and serial regions, a low CPU/Wall Clock ratio could indicate a performance bottleneck caused by one or more of the following:

  I/O calls—For example, `read`() or `write`() calls

  System calls—For example, `open`() and `close`() calls

  Memory accesses—For example, Cache misses

  Compare event metrics and latency for regions of interest to discover if the bottleneck is due to memory accesses.

Architecture-dependent metrics

The type of event metrics varies according to machine architecture. Available metric groups are based upon functionality. This section defines the groupings and the terms necessary to understand and interpret metrics and events you can collect using CXperf. The event groupings are:

Timer metrics	Wall Clock, CPU time, Execution counts
Process events	Context Switches (voluntary and involuntary), Migrations, Page Faults
Memory events	Data TLB misses, Instruction TLB misses, Cache misses, Instruction counts, Latency
Data Cache Utilization	Cache misses, Instruction counts, Latency
Data and Instruction TLB misses	Data TLB misses, Instruction TLB misses, and Instruction counts

The following sections define event metrics. The Timer metric set is the default set and is described in “Metrics available on all architectures” on page 43. Refer to the Glossary for additional terms and definitions to understand and interpret events metrics.

Process events

Process events are available on HP V-Class, K-Class, and D-Class servers. Process events are:

Context Switches	Occur when a process changes its state. The possible states for a process are *running*, *ready*, or *waiting/blocked*. Can be voluntary or involuntary (forced).
Migrations	Occur after a context switch when a process changes the CPU on which it runs.
Page Faults	Occur when a process requests data not currently in memory requiring the operating system to retrieve the page containing the requested data from disk.

Memory events

Memory events are available on the HP V-Class servers only and is not available on HP K-Class or D-Class servers. Memory events are:

Data TLB misses	Represent the number of times an address translation from virtual to physical memory was not found in the Translation Lookaside Buffer (TLB). In this case the address translation refers to data that is being referenced. The TLB is a cache of virtual-to-physical memory address translations for the most recently referenced page table entries.
Instruction TLB misses	Represent the number of times the address translation from virtual to physical memory for an instruction was not found in the TLB. The TLB is a cache of virtual-to-physical memory address translations for the most recently referenced page table entries.
Cache misses	Occur when data to be loaded is not residing in the cache.
Instruction counts (Inst)	Number of completed instructions.
Latency	Amount of time spent accessing memory to locate data or instructions not found in the processor's data or instruction cache.

Data Cache Utilization

Data Cache Utilization events are available on HP V-Class servers and not available on HP K-Class or D-Class servers. Data Cache Utilization events are:

Cache misses	Occur when data to be loaded is not residing in the cache.
Instruction TLB misses	Represent the number of times the address translation from virtual to physical memory for an instruction was not found in the TLB. The TLB is a cache of virtual-to-physical memory address translations for the most recently referenced page table entries.

Latency	Amount of time spent accessing memory to locate data or instructions not found in the processor's data or instruction cache. CXperf provides Data Cache Miss Latency (DCache Lat) and Instruction Cache Miss Latency (ICache Lat).

Data and Instruction TLB misses

Data and Instruction TLB miss events are available on HP V-Class, K-Class, and D-Class servers. TLB miss events are:

Data TLB misses	Represent the number of times the address translation from virtual to physical memory was not found in the TLB. In this case the address translation refers to data that is being referenced. The TLB is a cache of virtual-to-physical memory address translations for the most recently referenced page table entries. On SPP1600 Series systems the TLB contains 120 entries, on Exemplar S2000/X2000 and V-Class systems it contains 92 entries.
Instruction TLB misses	Represent the number of times the address translation from virtual to physical memory for an instruction was not found in the TLB. The TLB is a cache of virtual-to-physical memory address translations for the most recently referenced page table entries.
Instruction counts (Inst)	Number of completed instructions.

Derived metrics

During analysis, CXperf provides a number of metrics derived from the primitive metrics collected.

Although CXperf uses numbers accurate to four decimal places when calculating metrics, the values displayed in reports are rounded to two decimal places. When you use the rounded values from performance reports to calculate your own metrics, you cannot reproduce the values CXperf reports for derived metrics.

Refer to Chapter 6, "Analyzing," for more information about interpreting profiling data.

The following list defines derived metrics calculated by CXperf:

DTLB/Inst — Fraction of the total instruction counts for which the address translation from virtual to physical memory was not found in the TLB. In this case the address translation refers to data that is being referenced.

ITLB/Inst — Fraction of the total instruction counts for which the address translation from virtual to physical memory was not found in the TLB. In this case the address translation refers to instructions that are being referenced.

Event Latency/CPU — Ratio of time spent accessing memory to locate data not found in the processor's data cache to time spent computing with cached data.

metric/CPU — Ratio of any metric collected during a profiling session to the CPU time for that session. After analyzing the value of *metric* it is useful to consider this ratio. For example, it may be useful to normalize collected metrics in this way if the metric value is different when you compare different runs of the same process.

MIPS — Average MIPS (millions of instructions per second) is calculated during analysis if instruction counts, clock cycles, and Wall Clock time are collected. The formula CXperf uses to calculate average MIPS is:

$$\text{average\_MIPS} = \frac{\text{number\_of\_instructions\_completed}}{\text{wall\_clock\_time (sec)} \times (1 \times 10^{6})}$$

Using event metrics

Refer to "Metrics available on all architectures" on page 43 and "Architecture-dependent metrics" on page 44 for an outline of information provided by any metric. When you run an application under CXperf with regions and metrics selected for profiling, the code may execute more slowly than expected. This can be due to profiling intrusion (time delays) introduced by CXperf.

To obtain accurate profiling data, try to minimize the level of intrusion. Part of minimizing the intrusion is choosing metrics judiciously. Consider the following approach to profiling an application:

- Choose only CPU and Wall Clock time to collect for routines the first time you profile a program. The greater the number of regions and metrics you select for profiling, the greater the amount of profiling intrusion.

  CPU and Wall Clock times help identify routines that spend significant amounts of time waiting on memory. Once the routine times have been identified, you can further investigate specific routines and loops within those routines that consume the most CPU and Wall Clock time.

- Monitor events such as Cache misses and Latency for routines identified as problem routines. This helps identify reasons for poor performance, such as ineffective cache use or contended access to data among processors on the same hypernode.

- Compare and contrast metrics for different events. For example, if you observe a large number of memory miss events for a region, compare latency metrics for that region. If the average latency time is short, despite the large number of misses, then you might conclude that the total latency time for that region is not significant.

- Use the derived metrics during analysis. After analyzing the value of a particular metric, consider the ratio of `metric/CPU`. Normalized metrics can be useful if the metric value is different when you compare different runs of the same process.

Refer to Chapter 5, “Profiling,” for further discussion of profiling strategy.

Instrumenting

The second step in the performance analysis process after compilation is Instrumentation. Instrumentation can be divided into three tasks:

- Selecting regions to profile
- Selecting loop nesting level to profile
- Selecting metrics to collect

Profiling time and intrusion increase as the number of source code regions and metrics you choose to profile increases. Do not select all region types (routines, loops, and parallel loops) and all metrics in a single profiling session.

Ideally, region selection should proceed from coarse grained (routines) to fine grained (loops) as you identify code regions that exhibit performance problems.

The following sections describe how to perform each of the Instrumentation tasks in GUI mode or in line mode.

Instrumenting in GUI mode

Region types available for profiling with CXperf are routines, loops, and parallel loops. This section describes how to select specific regions to profile when you instrument a program in GUI mode.

Refer to Chapter 5, “Profiling,” for more information about profiling strategies.

Selecting routines and loops

If you start CXperf without the name of the executable file, CXperf first displays the Compilation Page. You can browse a file list and choose the file you want to profile from the Compilation Page. Refer to “Compiling” on page 32 for more information.

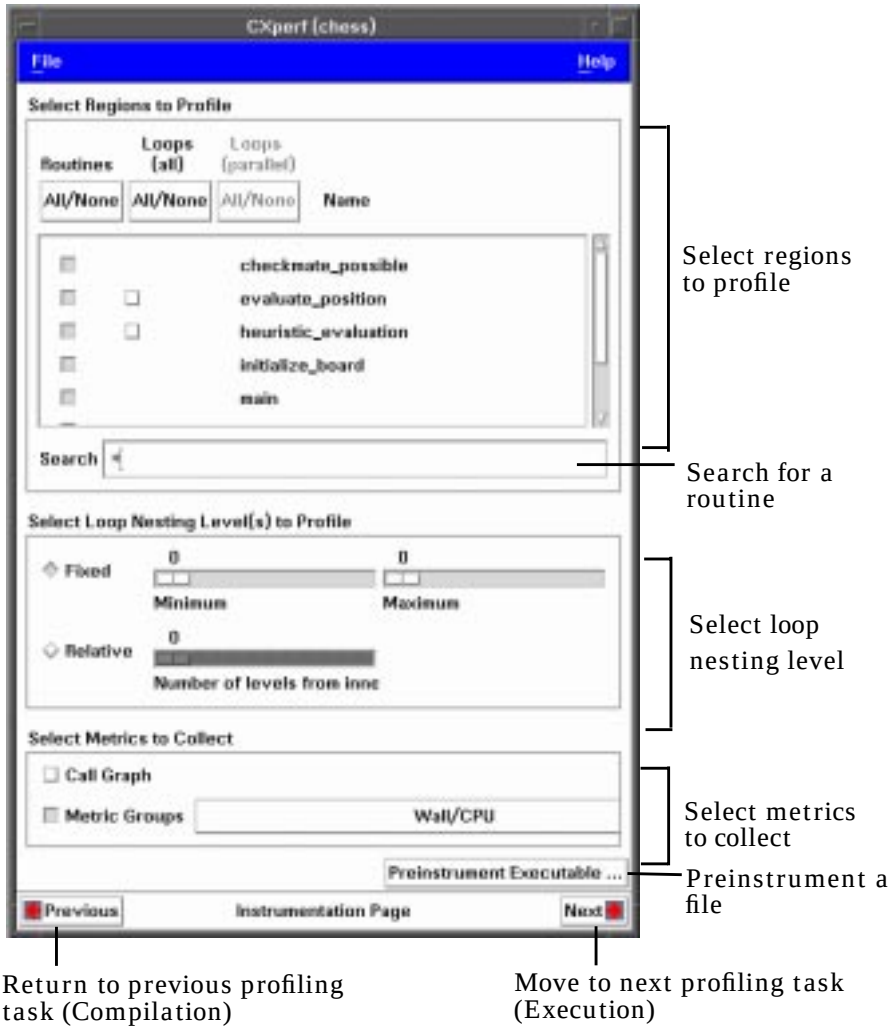
After you choose a program to profile, CXperf guides you to the next step, Instrumentation, by opening the Instrumentation Page.

When you invoke CXperf with the name of a correctly compiled executable file, for example,

```
% cxperf myexecutable.exe
```

CXperf displays the Instrumentation Page as shown in Figure 11.

Figure 11 Instrumentation Page



Use the Instrumentation Page to select or deselect source code regions to monitor during profiling. CXperf collects metrics at the regions you select. In GUI mode, all routines in your program are selected by default. You can change the default settings before you run a program.

You can select three region types for profiling:

Routines	Routines are available for profiling if you compiled the source code with HP compilers using the `+pa` or `+pal` option or if you used CXoi to instrument the program's archive libraries or object files.
Loops(all)	Loop regions are available for profiling if the source code contains loops and was compiled with HP compilers using the `+pal` option at optimization level `+O2` or `+O3`.
Loops(parallel)	Parallel Loops are compiler generated loops. Parallel Loop regions are available for profiling if you compiled your program with HP compilers using the `+pal` option at optimization level `+O3 +Oparallel`.

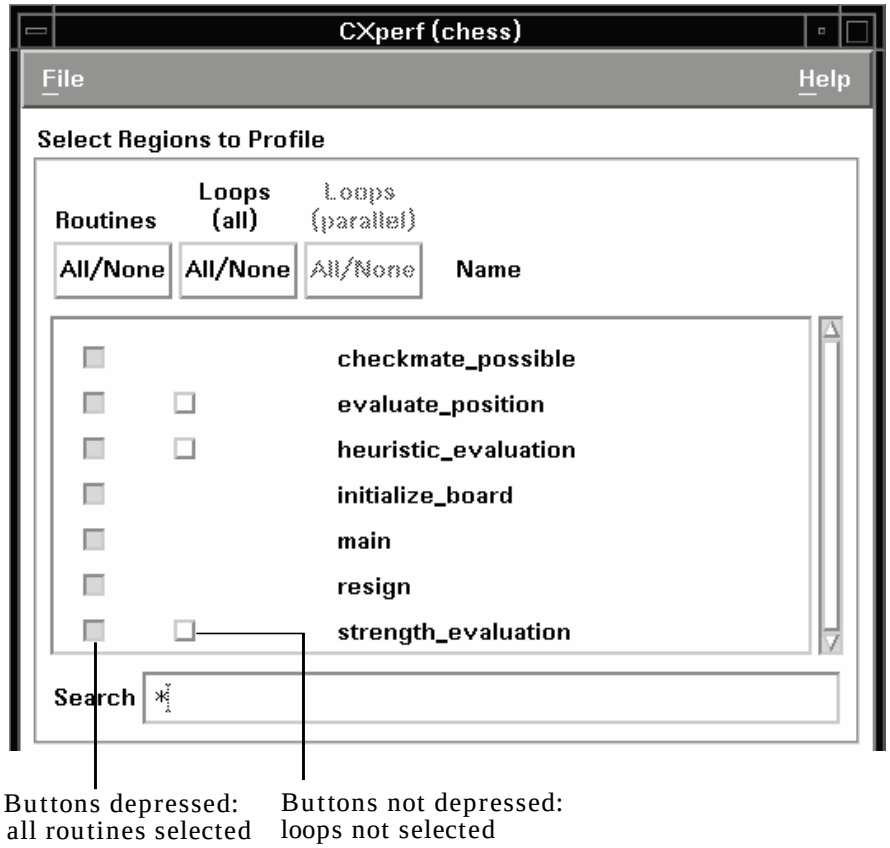
In GUI mode, if you compile your source code with the `+pa` or `+pal` option, all routines in the program are initially selected for profiling.

The default selections are different when you instrument your compiled program in line mode. Refer to "Instrumenting in line mode" on page 58 for details.

Use the top section of the Instrumentation Page to change the region types to profile.

The top section of the Instrumentation Page is shown in Figure 12. Use it to select regions to profile.

Figure 12 **Instrumentation Page: Select Regions to Profile**



Select routines or loops to profile using the buttons beside the routine names. Routine names are listed in alphabetical order. Use the **All/None** button to specify all routines, or use individual buttons to specify a subset of routines. If a button for a particular region type is not displayed for a routine, no region of that type is available for profiling for that routine. Metrics are collected at the source code regions selected in the specified set of routines.

For object files and archive libraries, only those routines that were instrumented for profiling with the CXoi utility can be profiled.

If your program contains a large number of routines and you need to search for a routine, the following options are available:

- Use the scrollbar to navigate the routine list.
- Type the name of the routine in the Search field. When you press **Return** the list scrolls so that the desired routine appears at the top of the list.
- Use wildcards in the name. Available wildcards are:
  - Question marks (?) to match single characters
  - Asterisks (*) to match multiple characters

  When you press **Return**, CXperf executes the search and, if it finds a match, displays the first matching routine name at the top of the list. Press **Return** again for more matching routines.
- Use the asterisks (*) alone in the Search field to display the top of the routine list.

Selecting loop nesting levels

If you choose to profile loops, you can specify:

- A fixed range of loop nesting levels, or
- The number of loop nesting levels to profile relative to the nest's innermost level.

Use the middle section of the Instrumentation Page to select a loop nesting level.

The middle section of the Instrumentation Page, demonstrating default loop nesting level settings, is shown in Figure 13.

Figure 13 Instrumentation Page: Default Loop Nesting Level

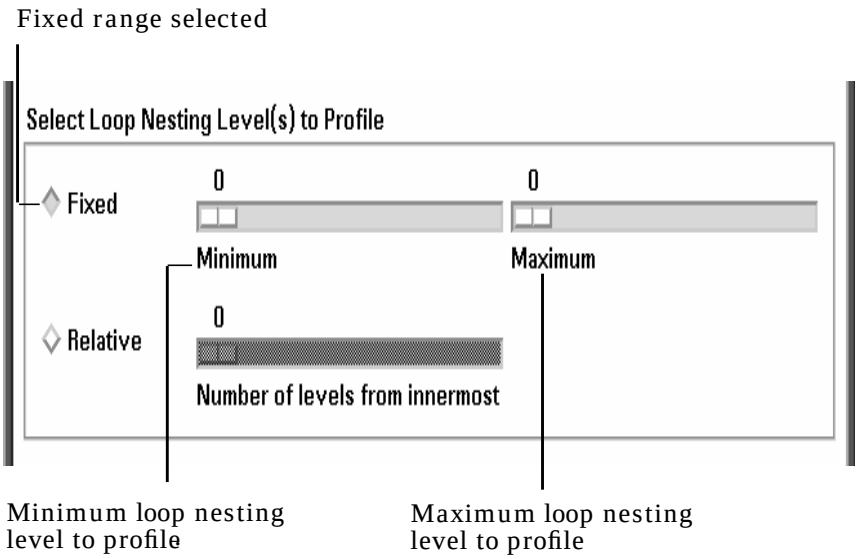


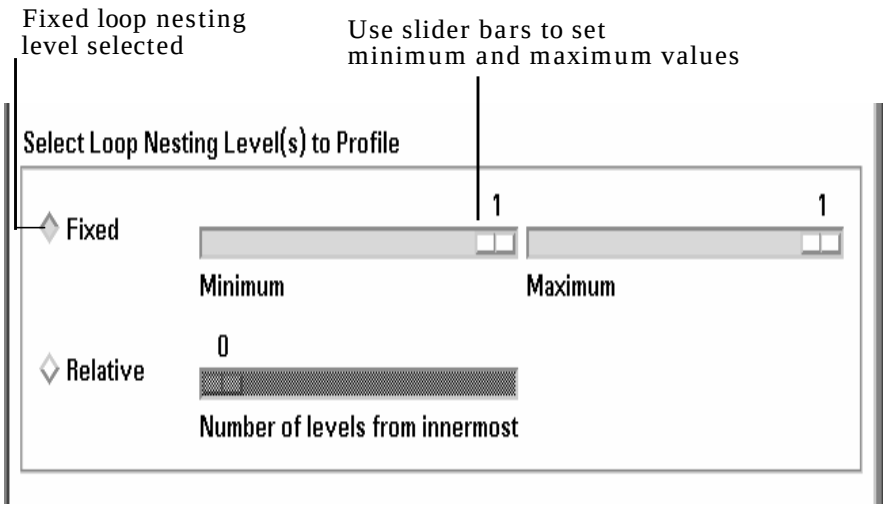
Figure 13 depicts default loop nesting level settings. The default setting specifies a fixed loop nesting level range with a minimum of 0 and a maximum of 1. All loops with a nesting level of 1 after optimization—outermost loops—are selected for profiling. This minimizes profiling intrusion. It is the recommended setting for an initial profiling session.

Setting a fixed loop nesting level range

On different runs of a program, you can select different sections or slices of the loops within the program for profiling. When specifying a fixed range of loop nesting, you should generally set the minimum loop nesting level equal to the maximum loop nesting level, as shown in Figure 14.

The middle section of the Instrumentation Page, demonstrating how to select fixed loop nesting level settings, is shown in Figure 14.

Figure 14 **Instrumentation Page: Select Fixed Loop Nesting Level**



Setting a relative loop nesting level

Instead of choosing a fixed range of loop nesting levels for profiling, you can specify the number of loop nesting levels to profile relative to the innermost loop nest of your program.

The meaning of relative loop nesting levels is as follows:

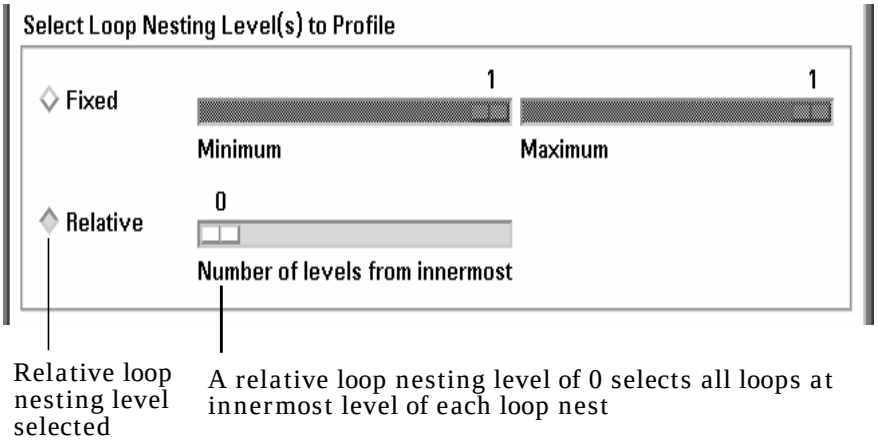
- A relative setting of 0 selects only the loops in the innermost (deepest) level of each loop nest.
- A relative setting of 1 selects only the loops in the innermost two nesting levels of each loop.

  For example, if the innermost nesting level of a loop nest is 4, and a relative setting of 1 is specified, the loops at nesting levels 3 and 4 of that loop nest are selected for profiling.
- A maximum setting, achieved by setting the slider bar as far right as possible, is equivalent to selecting all loops at all nesting levels.

When you specify a relative loop nesting level, loops that are not part of a loop nest are also selected for profiling.

The middle section of the Instrumentation Page, demonstrating how to select relative loop nesting level settings, is shown in Figure 15.

Figure 15 **Instrumentation Page: Select Relative Loop Nesting Level**

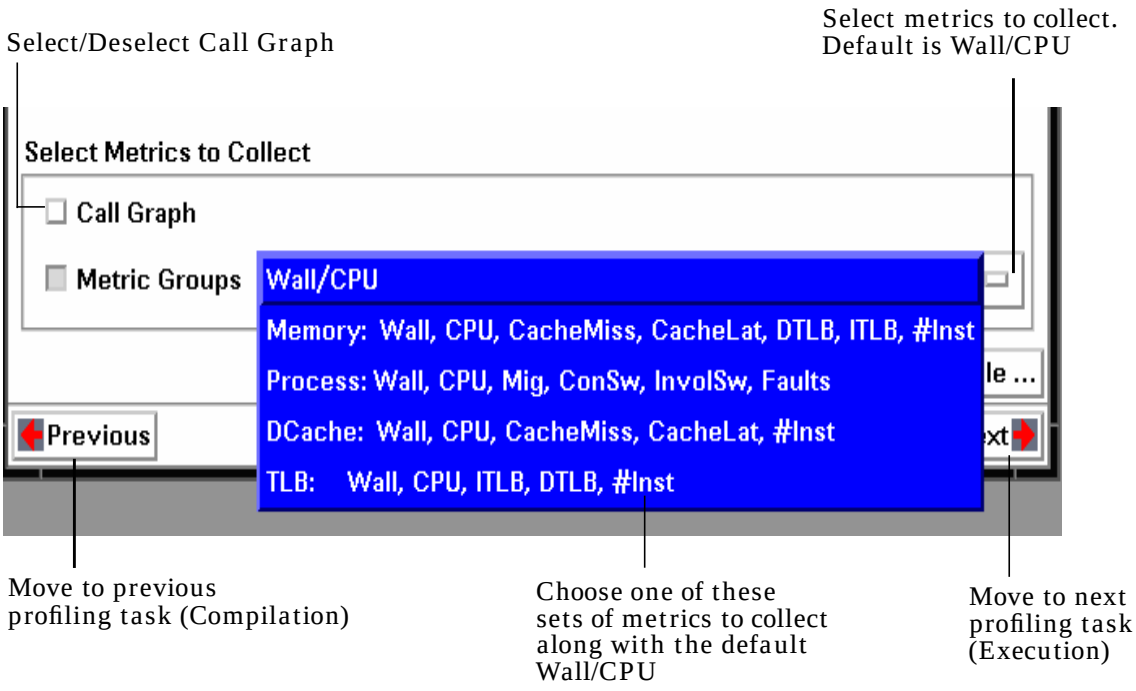


Selecting metrics to collect

The type of event metrics available varies according to machine architecture. Use the third and lowest section of the Instrumentation Page to select the metrics to collect. See Figure 16 for details of metric selection on the Instrumentation Page.

The bottom section of the Instrumentation Page is shown in Figure 16. Use it to select Call Graph and metrics to collect during profiling.

Figure 16 Instrumentation Page: Select Metrics to Collect


Select/Deselect Call Graph
Select metrics to collect. Default is Wall/CPU
Select Metrics to Collect
Call Graph
Metric Groups
Wall/CPU
Memory: Wall, CPU, CacheMiss, CacheLat, DTLB, ITLB, #Inst
Process: Wall, CPU, Mig, ConSw, InvolSw, Faults
DCache: Wall, CPU, CacheMiss, CacheLat, #Inst
TLB: Wall, CPU, ITLB, DTLB, #Inst
Previous
Move to previous profiling task (Compilation)
Choose one of these sets of metrics to collect along with the default Wall/CPU
Move to next profiling task (Execution)

The metric sets you can select for profiling appear in a pulldown menu. Available metric sets are different on different architectures. A metric set not displayed means that set is not available for profiling on the specified architecture. Wall clock and CPU time are always collected. You can choose to collect one additional metric set using the pulldown menu.

Use the Call Graph selection button to instrument a file so that Call Graph data is available during analysis.

Use the **Preinstrument Executable** button when you want to preinstrument an executable file. Refer to “Preinstrumenting” on page 65 for more details.

Instrumenting in line mode

Source code regions available for profiling with CXperf are routines, loops, and parallel loops. This section describes how to select specific regions to profile when you instrument a program in line mode.

Refer to Chapter 5, “Profiling,” for more information about profiling strategy.

Selecting routines and loops

When you invoke CXperf with the name of a correctly compiled executable file and the `-nw` option as shown in the following example:

```
% cxperf -nw myexecutable.exe
```

CXperf initially launches in line mode. Select or deselect region types for profiling using the `select` or `deselect` commands.

CAUTION **Do not rely on default region selections in line mode. No regions are initially selected. No metrics are collected if you run your program without invoking `select` to select regions to profile.**

You must specify the regions before you run a program. CXperf collects metrics in the selected regions. The following describes the region types available for profiling:

Routines	Routines are available for profiling if you compiled the source code with HP compilers using the `+pa` or `+pal` option or if you used CXoi to instrument the program's archive libraries or object files.
Loops(all)	Loop regions are available for profiling if the source code contains loops and was compiled with HP compilers using the `+pal` option at optimization level `+02` or `+03`.
Loops(parallel)	Parallel Loops are compiler generated loops. Parallel Loop regions are available for profiling if you compiled your program with HP compilers using the `+pal` option at optimization levels `+03` and `+Oparallel`.

The following sections describe the variants of `select` and `deselect`. Refer to the *CXperf Command Reference* or online help for more information about each command's syntax.

The current loop nesting level applies to any selection you make with the `select` command. Refer to "Selecting loop nesting levels" on page 61.

Selecting or deselecting one type of region in all routines

To select or deselect one type of region in all routines use the following syntax:

```
[ select | deselect ] [ routine | loop ] all
```

Use this command to:

- Select or deselect all routines in your program that were instrumented for profiling, or
- Select or deselect all instrumented loops (including parallel loops generated by HP parallel compilers) in all routines.

  Loop level profiling is only available for routines compiled with HP parallel compilers using the `+pal` option at optimization levels `+O2` or `+O3`.

Selecting or deselecting one region type in specific routines

To select or deselect one type of region in specific routines, use the following syntax:

`[ select | deselect ] loop in` *routine-list*

Use this command to select or deselect all instrumented loops in the specified routines.

Separate multiple routines in the list with a space. If two routines have the same name, prefix them with their file name followed by a colon:

file_name:routine_name

Loop level profiling is only available for routines compiled with HP compilers using the `+pal` option at optimization levels `+O2` or `+O3`.

Selecting or deselecting one region type at specific lines

To select or deselect one type of region at specific lines, use the following syntax:

`[ select | deselect ][ routine | loop ] at` *line_number_list*

Use this command to select or deselect instrumented routines or loops at the specified line numbers. The *line_number_list* specifies one or more line numbers that contain regions you want to select.

Separate multiple line numbers in the list with a space. To select a region that is not in the current file source file, prefix the line number with a file name followed by a colon as shown here:

file_name:*line_number*

For example the command:

`(CXperf)` **`select loop at calc.f: 3 15`**

selects the instrumented loops at lines 3 and 15 of the file calc.f, assuming they fall within the currently selected loop nesting level range. No other source code region selections are affected.

Use `list` to see source files and line numbers.

Loop level profiling is only available for routines compiled with HP compilers using the `+pal` option at optimization levels `+02` or `+03`.

Selecting or deselecting all regions in specific routines

To select or deselect all regions in specific routines, use the following syntax:

`[ select | deselect ]` *routine_name*

Use this command to select or deselect all instrumented regions of any type in the specified routines.

Separate multiple routine names in the list with a space. If two routines have the same name, prefix them with a file name followed by a colon:

file_name:*routine_name*

For example the command:

`(CXperf)` **`select file1:INIT CALC file2:INIT`**

selects all instrumented regions in routines INIT and CALC in file1, and instrumented regions in routine INIT in file2. No other source code region selections are affected.

Selecting loop nesting levels

When you select loops for profiling, by default only loops at nesting level 0 (after optimization) are selected. The default setting reduces the number of loops initially selected for profiling, thus minimizing the profiling intrusion incurred when profiling nested loops with large iteration counts.

Use `set visibility` to set the loop nesting level for profile data collection in line mode. The `loop_levels` parameter of `set visibility` allows you to specify either a fixed range of loop nesting levels to profile or a number of nesting levels relative to each loop nest's innermost level. Loop nesting level settings apply to all loop regions selected for profiling.

CXperf automatically determines the number of loop nesting levels in your program and sets the maximum loop nesting levels and the maximum number of levels from the innermost loop appropriately. These nesting levels correspond to the loops created by the compiler, and may not correspond directly to the original source code due to compiler optimizations.

If you choose to profile loops, you can specify:

- A fixed range of loop nesting levels, or
- The number of loop nesting levels to profile relative to the nest's innermost level.

Specifying a fixed loop nesting level range

To specify a fixed loop nesting level range use `set visibility` with the `loop_levels` parameter.

`set visibility loop_levels <`*min*`> <`*max*`>`

<*min*> and <*max*> are positive integers specifying the minimum and maximum loop nesting levels to profile, respectively. Separate the <*min*> and <*max*> values with a space. The default loop nesting level is `loop_levels 0 1`. A single entry is assumed to be the <*max*> value, and the optional <*min*> value defaults to 1.

NOTE If the <*min*> value is greater than the <*max*> value, then the values are reversed.

The first time you profile a program use the default `loop_levels`. On subsequent runs you can select different sections or slices of the loops within the program for profiling by specifying a minimum and maximum

loop nesting level. When you specify a fixed range of loop nesting level, set the minimum loop nesting level equal to the maximum loop nesting level.

For example the command:

```
(CXperf) set visibility loop_levels 1 1
```

selects loops only at nesting level 1 (after optimization) for profiling.

Specifying a relative loop nesting level

To specify a relative loop nesting level use `set visibility` with the `loop_levels innermost` parameter.

`set visibility loop_levels innermost` *num_levels*

This specifies the number of loop nesting levels to profile relative to the innermost loop of each loop nest in your program.

The meaning of relative loop nesting levels is as follows:

- A relative setting of 0 means only the loops at the innermost (deepest) level of each loop nest are selected for profiling.
- A relative setting of 1 selects only the loops at the two innermost nesting levels of each loop nest for profiling.

  For example, if the innermost nesting level of a loop nest is 4 and you specify a relative setting of 1, the loops at nesting levels 3 and 4 are selected for profiling.
- A maximum setting selects all loops at all loop nesting levels.

When you specify a relative loop nesting level, loops that are not part of a loop nest are also selected for profiling. When you specify a relative loop nesting level setting, you must use the `innermost` keyword with the `loop_levels` parameter of `set visibility`.

For example the command:

```
(CXperf) set visibility loop_levels innermost 2
```

selects the three innermost nesting levels of each loop for profiling. If you do not specify the number of levels with the `innermost` keyword, CXperf assumes the default value of 0, and only the innermost loop of any nests is selected for profiling. Any loops that are not part of a loop nest are also selected for profiling.

Selecting metrics to collect

The type of event metrics available varies according to machine architecture.

Specify metrics to collect during profiling. Use `collect` followed by `set events`.

The following example demonstrates how you can use these commands in conjunction with other CXperf commands:

```
(CXperf) select all
(CXperf) collect cpu wall_clock call_graph events
(CXperf) set events memory
```

In this example:

- `select all` selects all routines and loops in your program for profiling. Refer to "Selecting routines and loops" on page 58 for details about `select`.
- `collect` specifies that CPU, Wall Clock, Call Graph, and events are to be collected.

  CPU and Wall Clock times are default metrics, and are always collected. You must include `call_graph` in the `collect` command when you want to analyze Call Graph data.

  `events` are architecture-specific metrics. You must further specify `events` with the `set events` command
- `set events` specifies the type of `event` to collect. In this example, collect Memory events.

  You can collect one set of events per program run using the `set events` command.

The options available for the `set events` command depend on the metric groups available for the specified architecture. Refer to "Metrics available on all architectures" on page 43 and "Architecture-dependent metrics" on page 44 for details.

Use the following syntax for the `set events` command:

```
set events { memory | process | tlb_misses | data_cache }
```

The `set events` options map to the following available metric groups:

Memory events	Data TLB misses, instruction TLB misses, cache misses, instruction counts, latency. Memory events are available on HP V-Class, K-Class, and D-Class servers.
Process events	Context switches (voluntary and involuntary), migrations, page faults. Process events are available on HP V-Class, K-Class, and D-Class servers.
Data and Instruction TLB misses	Data TLB misses, instruction TLB misses, and instruction counts. Data and instruction TLB misses are available on HP V-Class, K-Class, and D-Class servers.
Data Cache Utilization	Cache misses, instruction counts, latency. Data cache utilization events are available on HP V-Class servers and not available on HP K-Class or D-Class servers.

Refer to “Introducing metrics” on page 42 for metric definitions and details.

Preinstrumenting

You can write profile selection settings (instrumentation) to the current executable file or to a copy of the current executable file. This is preinstrumenting an executable file. You can run the resulting file outside the control of CXperf and collect profiling data in a performance data file (PDF) for later analysis. You can run the preinstrumented file under the control of CXperf by invoking CXperf with the name of the preinstrumented executable file.

Use preinstrumented executable files to:

- Profile in environments that do not support CXperf controlling a child process.
- Profile applications in conjunction with tools such as MPI or PVM that replicate processes. For more information refer to “Profiling MPI and PVM applications” on page 79.
- Profile applications where a driver program or script starts the process. For more information refer to “Batch mode” on page 85.
- Maintain separate copies of an executable file with different regions and metrics selected for profiling. Doing this makes it easy to generate multiple PDFs for comparison and analysis.

Setting the environment

Preinstrumentation in CXperf is a powerful function. This section provides information to allow you to maximize this functionality.

Performance Data Files (PDFs)

When you run a preinstrumented executable file outside the control of CXperf, the profiling data is collected in a PDF for later analysis. The PDF is named *executable.pid*.pdf. The *pid* is the program’s HP-UX process ID.

CXperf command line options

When you preinstrument an executable file on one architecture and run it on a different architecture to generate profiling data, specify `-tm` <*architecture*> when you start CXperf to preinstrument. This calls the correct timing routines to collect metrics for the target system.

Valid values for architecture are described in Table 5.

Table 5 **-tm <*architecture*>: valid values**

<*architecture*>	target system
v-class	HP V-Class
hp700	HP D- or K-Class, 700 series models.
hp800	HP D- or K-Class, 800 series models.

For example, if you run CXperf on an HP K-Class to preinstrument an executable file that will be run on an HP V-Class to collect profiling data, start CXperf as follows:

```
% /opt/cxperf/bin/cxperf -tm v-class
```

For more details about command line options to invoke CXperf refer to `cxperf` in the *CXperf Command Reference* or type `cxperf -help` at your UNIX prompt.

The PROFDIR environment variable

Set the PROFDIR environment variable to write PDFs created by a preinstrumented program to a predetermined directory. The directory must exist and you must have write permissions.

If PROFDIR does not exist, CXperf creates *executable.pid*.pdf in the directory the application completes execution (usually the directory from which the application is invoked). *executable* is the name of the executable file and *pid* is the program's HP-UX process ID.

If the PROFDIR environment variable is set as follows:

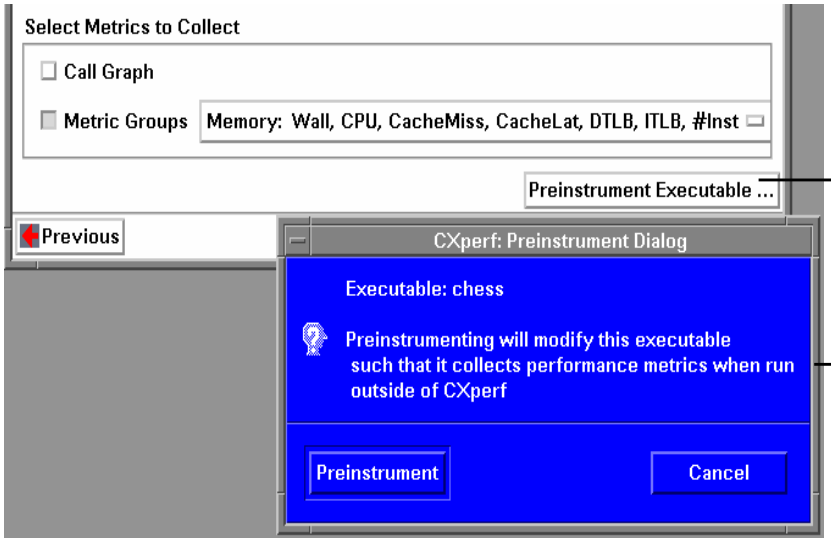
PROFDIR = *path*

path/*executable.pid*.pdf is the path and name of the PDF, where *pid* is the program's HP-UX process ID. If the PROFDIR variable is an empty string, no PDF is created.

Preinstrumenting in GUI mode

To preinstrument a file in GUI mode, start by selecting regions, loop nesting, and metrics on the Instrumentation Page. Save the selections to the file using the **Preinstrument Executable** button on the bottom of the Instrumentation Page. Figure 17 indicates the position of the **Preinstrument Executable** button and demonstrates the dialog that appears after you click the **Preinstrument Executable** button.

Figure 17 Instrumentation Page: Preinstrument Executable


Select Metrics to Collect
Call Graph
Metric Groups
Memory: Wall, CPU, CacheMiss, CacheLat, DTLB, ITLB, #Inst
Preinstrument Executable ...
Previous
CXperf: Preinstrument Dialog
Executable: chess
Preinstrumenting will modify this executable such that it collects performance metrics when run outside of CXperf
Preinstrument
Cancel
Use the Preinstrument Executable button to preinstrument your program
The Preinstrument dialog prompts you to confirm you want to complete preinstrumentation for a particular executable file

When you preinstrument an executable file, it gets modified so that it collects performance data when you run it outside of CXperf. This section describes the procedure to preinstrument a file in GUI mode.

Step 1. Compile the program.

Step 2. Start CXperf with the executable file.

Step 3. Select profiling data from the Instrumentation Page.

Select the regions you want to profile, the loop nesting level and the metrics you want to collect as you would if you were going to run your application under the control of CXperf. Refer to “Instrumenting in GUI mode” on page 49 for details.

Step 4. Save the preinstrumented executable file.

Use **Preinstrument Executable** on the bottom of the Instrumentation Page. This saves the executable file modified with the profile selection settings.

Step 5. Run the preinstrumented file outside the control of CXperf.

CXperf collects profiling data in a PDF for later analysis.

CXperf names the PDF *executable.pid*.pdf, where *executable* is the executable file and *pid* is the program's HP-UX process ID.

For example, if you preinstrument your program and save the preinstrumented program as a.out.inst, you can run the executable from the shell to generate a PDF as follows:

```
% a.out.inst
```

CXperf names the PDF *executable.pid*.pdf. The name might be a.out.inst.1234.pdf where 1234 is the program's HP-UX process ID.

By default, the PDF is created in the directory where the application completes execution, usually the directory from which the application is invoked. Use the environment variable PROFDIR to change the directory where the PDF is created. Refer to "The PROFDIR environment variable" on page 66 for further details.

Step 6. Invoke CXperf with the name of the PDF.

For example, using the PDF created in Step 5, use the following command:

```
% /opt/cxperf/bin/cxperf a.out.inst.1324.pdf
```

The Analysis Page appears. Refer to Chapter 6, "Analyzing," for details about analyzing profiling data.

Preinstrumenting in line mode

To preinstrument a file in line mode, start by selecting the regions, loop nesting, and metrics you want using tty commands. Use `save executable` to write the instrumentation to the executable file or to a copy of the executable file

Step 1. Compile your program.

Step 2. Start CXperf with the executable file.

Step 3. Select profiling options.

Select the regions you want to profile, the loop nesting level, and the metrics you want to collect as you would if you were going to run your application under the control of CXperf. Refer to “Instrumenting in line mode” on page 58 for details.

Step 4. Save the preinstrumented executable file.

Use `save executable` to write the instrumentation to the executable file or to a copy of the executable file. Options are:

- Execute `save executable` without specifying a file name, and CXperf writes the instrumentation to the current executable file without changing its name.
- Execute `save executable` specifying a file name, and CXperf writes the instrumentation to a copy the current executable file, using the specified file name.

Step 5. Run the executable file outside CXperf's control to create a PDF.

CXperf collects profiling data in a PDF for later analysis. It names the PDF *executable.pid*.pdf, where PID is the program's HP-UX process ID.

For example, if you preinstrument your program and save the preinstrumented program as a.out.inst, you can run the executable from the shell to generate a PDF as follows:

```
% a.out.inst
```

CXperf names the PDF *executable.pid*.pdf. The name might be a.out.inst.1234.pdf where 1234 is the program's HP-UX process ID.

By default, the PDF is created in the directory where the application completes execution, usually the directory from which the application is invoked. Use the environment variable PROFDIR to change the directory where the PDF is created. Refer to “The PROFDIR environment variable” on page 66 for further details.

Step 6. Invoke CXperf with the name of the PDF, using line mode or GUI mode.

To analyze the PDF created in Step 5 in line mode enter:

```
% /opt/cxperf/bin/cxperf -nw a.out.inst.1324.pdf
```

```
% analyze
```

In this example, `analyze` creates a performance analysis report. A partial example output is shown below.

```
CXperf Version 6.0 Profile
Executable      : /test/cxperf_red/example
Profile Data    : /test/cxperf_red/example.pdf
Process State   : exited
CPU Time        :   1.525
Wall Clock Time : 232418724.000
Architecture    : HP9000/800 (4 threads)
=====================================================================
=====================================================================
          Routine Performance Analysis (Whole Application)
=====================================================================
 Call Counts
Count PS Routine Name
-------- -- ------------
       2    show_grades
 .....lines of output deleted.....
```

Refer to Chapter 6, “Analyzing,” for details about analyzing profiling data in line mode.

Even when you use line mode or batch mode to profile your application, you can analyze the PDF in GUI mode to make use of graphical analysis functionality.

To analyze the PDF created in Step 5 in GUI mode enter:

```
% /opt/cxperf/bin/cxperf a.out.inst.1324.pdf
```

The Analysis Page appears. Refer to Chapter 6, “Analyzing,” for details about analyzing profiling data in GUI mode.

5 Profiling

In this chapter you learn general profiling strategies to optimize the collection and analysis of performance data. This chapter also describes how to profile message passing applications. You learn how to use the PDF files CXperf generates and how to use CXperf in batch mode. Topics covered include:

- Profiling strategy
  - Profiling intrusion
  - Minimizing intrusion
  - Routines that call uninstrumented routines
- Profiling MPI and PVM applications
  - Generating PDFs
  - Using CXmerge
- Using Performance Data Files (PDFs)
  - Invoking CXperf with a PDF
  - Changing PDFs during a CXperf session
- Batch mode
  - Using a command file
  - Using a script

Profiling strategy

When you run an application under the control of CXperf with regions and metrics selected for profiling, your code may execute slower than expected. This can be due to profiling intrusion (time delays) introduced by CXperf. To obtain more accurate profiling data, you should minimize the amount of instrumentation used to collect the data.

This section describes the causes and effects of profiling intrusion. It provides a profiling strategy that helps quickly locate source code regions with performance problems.

Profiling intrusion

All methods of profiling are intrusive. The overhead associated with collecting profiling data can affect the validity of the results.

The more regions and metrics you select for profiling during a profiling run, the greater the intrusion introduced. One result of this intrusion is longer run times.

Time delays occur when CXperf accesses hardware counters that provide metric data at data sampling points. Each source code region enabled for profiling has a minimum of two data sampling points—a region entry point and a region exit point. The more sampling points, the greater the amount of profiling intrusion. Time delays also occur when CXperf stores profiling data during a program run. The more data CXperf must store, the greater the intrusion.

When only routines are selected for profiling, profiling intrusion is minimal. Loop profiling is more intrusive because the number of data points CXperf samples is far greater, especially in loop nests or in loops with large iteration counts.

By default, CXperf profiles loop nesting level 0—outermost loops—only. You can change the loop nesting level setting on the Instrumentation Page in GUI mode or using `set visibility` in line mode. Refer to “Instrumenting” on page 49 for more details.

The following example uses a simplified source code region structure to demonstrate how profiling intrusion can occur:

```
ROUTINE CALLED n TIMES
100 ITERATIONS OF LOOP AT NESTING LEVEL 0
100 ITERATIONS OF LOOP AT NESTING LEVEL 1
100 ITERATIONS OF LOOP AT NESTING LEVEL 2
```

There is an increase in the number of data sampling points as you select more loops for profiling. The relationship between the number of data sampling points enabled and the region type selected is shown in Table 6.

Table 6 Intrusion for loop profiling

Region types selected for profiling	Number of sampled data points
Routines only	2*n
All loops at nesting level 0 only	(100*2)*n = 200*n
All loops at nesting level 1 only	(100*2)*n = 200*n
All loops at nesting level 2 only	(100*2)*n = 200*n
All loops at all nesting levels	(100*2)*(100*2)*(100*2)*n = 8,000,000*n
All routines, all loops, and all nesting levels.	(2*n) + (8,000,000*n)
n is the number of times the routine was called	

The number of sampled data points in a loop nest grows by twice the number of iterations of a loop nest with each level of nesting. As illustrated in Table 6, profiling all loops at all nesting levels or profiling all region types during a single program run results in large numbers of sampled data points, which in turn increases the profiling intrusion.

Minimizing intrusion

Consider two key principles when you are profiling:

- Minimize the number of regions and metrics you select for profiling during each run of your program to reduce intrusion and improve the validity of the profiling data.
- Select region types from coarse-grained (routines) to fine-grained (loops or parallel loops) as you identify regions that exhibit performance problems.

The following procedure outlines a profiling strategy to reduce intrusion and time delays caused by CXperf collecting metric data. This is a top-down strategy, profiling routines first and then loops.

Step 1. Profile only routines and collect only Timer metrics the first time you profile your program with CXperf.

- Select all routines (or fewer, if you can already identify critical routines) for profiling.
- Collect Timer metrics. These default metrics include CPU time, wall clock time, and execution counts, and are available on all supported HP servers.

Doing this provides an overall, coarse view of your program's performance. Identify the routines that take the longest to execute.

Step 2. Rerun your program under CXperf to profile only critical routines whose performance you want to improve.

From the critical routines, select loops at loop nesting level 0. This section or slice of the loops contains only the outermost loops. Continue to collect only CPU and Wall Clock time.

Doing this provides a loop-level view of the routines without incurring the intrusion associated with selecting all loops at all nesting levels.

Step 3. Profile different sections or slices of the loops within the critical routines.

Rerun your program under CXperf control and select different sections or slices of the loops than the ones selected in Step 2. To change your loop nesting level settings in GUI and line mode, refer to "Selecting loop nesting levels" on page 53 and "Selecting loop nesting levels" on page 61 respectively.

Step 4. Collect different metrics at the regions and loops you identified with performance problems.

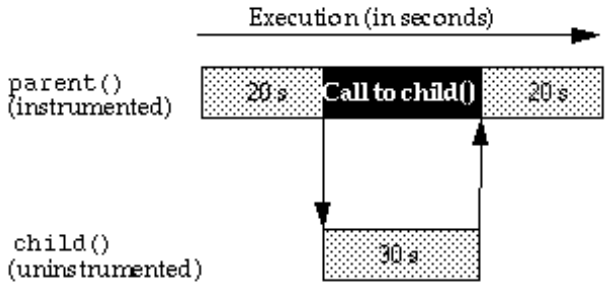
After you identify loops that are causing performance problems, collect and compare different metrics at those regions. With fewer regions selected, CXperf spends less time accessing the timing routines it uses to collect data. As a result the profiling data is more accurate.

Refer to "Introducing metrics" on page 42 for details about the metrics available on different architectures.

Routines that call uninstrumented routines

If an instrumented routine calls an uninstrumented routine, CXperf cannot separate the time spent in the uninstrumented child routine from the time spent in the instrumented parent. Figure 18 illustrates the condition.

Figure 18 Uninstrumented child processes



In Figure 18, routine `parent()` is instrumented for profiling and `child()` is not. The time spent in `parent()`, not including children, is reported as 70 seconds because CXperf cannot separate time spent in `child()`.

If routine `child()` is instrumented for profiling, CXperf correctly reports the time spent in `parent()`, not including children, as 40 seconds.

The greater the number of region types and metrics selected for profiling during a program run, the greater the amount of profiling intrusion introduced, and the greater the time delays. “Minimizing intrusion” on page 76 suggests that you select at most all routines for profiling and select fewer if you already identified the critical routines.

However, selecting fewer than all routines in your program effects the interpretation of a Call Graph in a fashion similar to that outlined in Figure 18.

If a routine is not selected for profiling during Instrumentation, be aware of the following features when interpreting your Call Graph:

- The non selected routine does not appear as a node on a Call Graph.
- An arrow directly connects the routine that called the omitted routine to the routines called by the omitted routine.
- Metric data that should be attributed to the omitted routine is attributed to the routine that called the omitted routine.

Profiling MPI and PVM applications

CXperf allows you to simultaneously profile all the processes generated by a Message Passing Interface (MPI) or Parallel Virtual Machine (PVM) application. CXperf generates a separate performance data file (PDF) for each of the application's processes. To analyze the application, combine the separate PDFs into a single PDF using the CXmerge utility. For more information about CXmerge refer to "Using CXmerge" on page 80.

Generating PDFs

To generate profiling data from an MPI or PVM application, perform the following steps:

Step 1. Compile.

Prepare the application for profiling with CXperf by compiling with the appropriate options. For more information, refer to "Compiling" on page 32.

Step 2. Preinstrument.

Select the metrics you want to collect and the regions to profile. When you run the program, the selected instrumentation applies to all of the processes generated by the application.

Refer to "Preinstrumenting" on page 65 for more information about preinstrumenting your application.

Step 3. Quit CXperf.

Step 4. Run the application from the shell.

Your application is not under the control of CXperf, but profiling instructions were written to your application in Step 2. Profiling data is collected when you run your program outside of CXperf.

CXperf generates a separate PDF for each of the application's processes. The appropriate process ID (PID) is inserted into the name of the PDF, uniquely naming each PDF using the format *executable.pid*.pdf.

If the PROFDIR environment variable does not exist or is not set, CXperf creates *executable.pid*.pdf in the directory in which the application completes execution (usually the directory from which the application is invoked). *executable* is the name of the executable file and *pid* is the program's HP-UX process ID.

Set the PROFDIR environment variable to write PDFs created by a preinstrumented program to a predetermined directory. The directory must exist and you must have write permissions.

For example, if the PROFDIR environmental variable is set as follows:

PROFDIR = *path*

path/*executable.pid*.pdf is the path and name of the PDF, where *pid* is the program's HP-UX process ID.

If the PROFDIR variable is an empty string no PDF is created.

Using CXmerge

To analyze the profiling data collected for an MPI or PVM application, merge individual PDFs into a single PDF using the CXmerge utility. You can use CXmerge to merge a number of PDFs created with the same version of CXperf. CXmerge is a separate utility shipped with CXperf. Refer to the cxmerge(1) man page for more information.

Syntax

To merge a number of separate PDFs, use the following syntax:

`cxmerge [-v...] -o` *output_data_file base_data_file* `[`*data_file*`]`

where

-v	Specifies verbose output. May be specified multiple times.
-o	Specifies output file. Merged data is written to output_data_file.
output_data_file	Specifies the file to which output data is written.
base_data_file	Specifies the executable file all other file's PDFs must match. All files to be merged must come from the same executable file with the same instrumentation selections.

data_file Specifies other data files to merge with the base data file.

All files to be merged must come from the same executable file with the same instrumentation selections.

For example, the following PDFs were created by running the preinstrumented MPI executable file, mpijob, outside CXperf control:

- mpijob.1000.pdf
- mpijob.1001.pdf
- mpijob.1002.pdf

To merge the three PDFs into a single PDF called merge.pdf, use the following command:

```
% cxmerge -o merge.pdf mpijob.1000.pdf
mpijob.1001.pdf mpijob.1002.pdf
```

Analyzing merged data

After the merged PDF is generated, use the following procedure to analyze the profiling data:

Step 1. Start CXperf specifying the name of the PDF generated by the `cxmerge` command.

- In GUI mode use the following syntax:

  `cxperf` *filename*`.pdf`

- In line mode use the following syntax:

  `cxperf -nw` *filename*`.pdf`

where

`-nw` Specifies the no windows option, starting CXperf in line mode.

filename`.pdf` Specifies the name of the PDF generated when you merge separate PDFs

Step 2. Analyze the merged data.

- When you invoke CXperf in GUI mode, the Analysis Page appears. Use the functionality on the Analysis Page to examine the profiling data.
- When you invoke CXperf in line mode, use `analyze` at the command prompt as shown here.

  (CXperf) **`analyze`**

Text reports are available in both GUI and line mode. A Summary Profile and Parallel Profile are available only if you analyze in GUI mode.

The following profile information helps you interpret Summary and Parallel Profiles for merged PDFs:

- **Summary Profile**—The profiling data in the Summary Profile represents the sum of the data for each region type across all processes, except for Wall Clock time. For Wall Clock time, the bars on the graph represent the maximum amount of Wall Clock time spent in each routine across all processes.
- **Parallel Profile**—The profiling data for each process in the Parallel Profile is mapped to the thread axis. Each bar on the graph represents the total time by region type spent in a single process.

Using Performance Data Files (PDFs)

When you profile a program, CXperf generates a performance data file (PDF) to store the profiling data. The PDF is a binary file containing performance data for a single run of your program. Performance analysis reports and graphs are generated from data in PDFs.

You can invoke CXperf with the name of a PDF to analyze data collected during a single run of a program or to analyze data collected in multiple PDFs but merged into a single PDF. For more details about merged PDFs refer to “Using CXmerge” on page 80.

Invoking CXperf with a PDF

When you invoke CXperf in GUI mode with the name of a PDF, CXperf opens onto the Analysis Page. You can analyze the data in that PDF using the functionality on the Analysis Page.

Refer to Chapter 6, “Analyzing” for details of Analysis Page functionality.

You can invoke CXperf in line mode with the name of a PDF and then use `analyze` to analyze profiling data.

In line mode, use `set pdf` during a CXperf session to specify the name of a PDF to be written or read. Refer to the *CXperf Command Reference* for more information about the `analyze` and `set pdf` commands.

Changing PDFs during a CXperf session

You can change the PDF to be written or read during a CXperf session. You may want to do this for two reasons:

- To prevent CXperf from overwriting an existing PDF.

  CXperf generates a PDF using the default name *executable*.pdf when you invoke CXperf with the name of an executable file and run your program. If you rerun the same executable file under CXperf control, CXperf overwrites all data in the original *executable*.pdf unless you change the name of the PDF.

  In GUI mode, change the name of the PDF between runs of your program.

In line mode or batch mode, use `set pdf` to change the name of the PDF between runs of your program. For example:

```
(CXperf) set pdf /usr/data/new.pdf
```

sets the name of the PDF to new.pdf. If a program is run, performance data is collected in the file /usr/data/new.pdf. To generate a report,CXperf analyzes the data in /usr/data/new.pdf.

- To analyze a different PDF.

  You can analyze and compare data for several PDFs during a single CXperf session. The following describes how to analyze multiple PDFs or PDFs created on different architectures or from different executable files:

  In GUI mode, use the Tear Off Analysis function on the Analysis Page to create an additional fully functional Analysis Page. You can analyze multiple PDFs at the same time. To open other PDFs you can either:

  –Tear off the current Analysis Page by using the Tear Off Analysis functionality.

  –Use Open File from the File menu.

  In line mode or batch mode, use `set pdf` before `analyze` to select a new PDF during a profiling session as shown here:

  ```
  (CXperf) set pdf /usr/data/other.pdf
  (CXperf) analyze
  ```

  In the example above, `set pdf` sets the name of the PDF to other.pdf. `analyze` then reads other.pdf to generate reports.

Batch mode

You can make use of CXperf's line mode commands to profile applications in batch mode. This section describes how to use CXperf in batch mode from the command line and from a shell script.

Using a command file

A command file is a text file that contains a list of CXperf commands. Use the command file to provide a batch of commands to CXperf.

The following syntax shows how to invoke CXperf to execute a command file at startup, read input to your program from a file, and redirect output and messages to a file:

`cxperf -x` *cmdfile* `a.out <` *input_file* `>&` *output_file*

where:

`-x`	Specifies the command file.
cmdfile	Command file.
input_file	Specifies the file to read input to your program.
output_file	Specifies the file to direct output and messages.

Command file input using the -x option

Use the -x option on the command line to execute CXperf in batch mode. CXperf executes the command file specified with the -x option.

A command file contains a list of CXperf commands. Each command must appear on a separate line. The # symbol denotes a comment.

The following is an example of a CXperf command file.

```
#This line is a comment.
#This is a CXperf command file to collect CPU and Wall Clock
#time for all routines and store the output in a file named
#CXperf.report.
select routine all
collect cpu wall_clock
run
analyze > CXperf.report
quit
```

For example, when you execute the command:

```
% cxperf -x cmdfile a.out
```

CXperf executes the command file (cmdfile) and quits when it encounters the `quit` command or the end of file (EOF).

Argument input using the -e option

Use the -e option on the command line to specify arguments to the program you are profiling. The arguments are used when you execute your program with the `run` command. For example:

```
% cxperf -x cmdfile -e a.out 12 35 14
```

CXperf executes the command file (cmdfile) and quits when it encounters the `quit` command or the end of file (EOF). CXperf expects the name of the executable file followed by program arguments to follow the -e option. No other CXperf options may follow the -e option.

Using a script

CXperf line mode commands and command files can be incorporated into shell scripts. To use CXperf in batch mode from a script do the following:

- Integrate CXperf commands into a script.
- Invoke the script with the -`profile` option.

The following example demonstrates integrating CXperf into a script that compiles and runs a program. This examples assumes the use of HP parallel compilers.

```
#!/bin/csh -f
#Name: batch_script
#Run this script with CXperf if command line option -profile
#is found
set PROFILER = ‘ ‘
set PROFILER_COMP_FLAG = ‘ ‘
for each arg ($argv)
 if ($arg == ‘-profile’) then
   set PROFILER = ‘/opt/cxperf/bin/cxperf -x cmdfile -e’
   set PROFILER_COMP_FLAG = ‘+pa’
   cat << EOF >! cmdfile
    select routine all
    collect cpu wall_clock
    run
    analyze
    quit
EOF
 endif
end
#compile a.out
/opt/ansic/bin/c89 $PROFILER_COMP_FLAG +O0 foo.c -o a.out
#Run the executable
$PROFILER a.out arg1 arg2
```

To profile with CXperf in batch mode using this script, use the following command line:

```
% batch_script -profile
```

The -profile option specifies that this script is run with CXperf. This script compiles the program with the +pa option, invokes CXperf with the resulting executable file, and executes a CXperf command file that performs a batch profiling session.

6 Analyzing

In this chapter you learn about analyzing profiling data in both GUI and line mode. You become familiar with text and graphical reports and the features available to configure reports. Topics covered include:

- Analysis Page
  - Toolbar
  - Configuration options
- Graphical Analysis
  - Accessing profiling data
  - Summary Profile
  - Parallel Profile
  - Call Graph
- Text Reports
  - Accessing profiling data in GUI mode
  - Accessing profiling data in line mode
  - Report fields
  - Summary and Parallel Reports
  - Call Graph Report
  - Line Mode Report

Analysis Page

When you run a program in GUI mode, CXperf guides you to the Analysis Page where performance data displays. Think of the Analysis Page as home base for your analysis in GUI mode.

To analyze data in line mode, refer to “Accessing profiling data in line mode” on page 112 for details. You can profile an application in line mode and still make use of the Analysis Page in GUI mode to analyze the Performance Data File (PDF). Invoke CXperf with the name of the PDF created in the line mode profiling session.

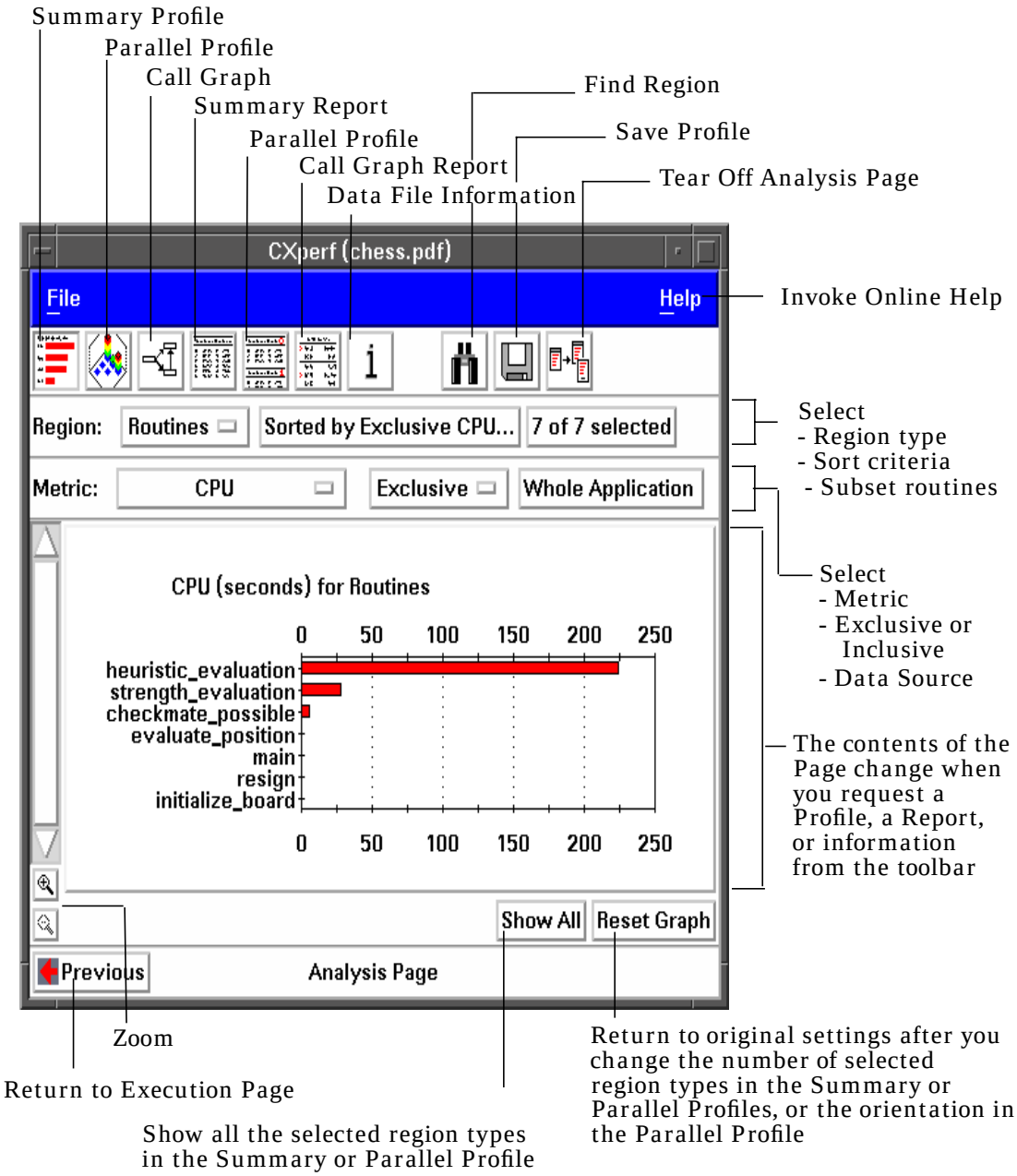
The Analysis Page has a toolbar menu and pulldown menus that allow you to select different types of data analysis and other options. The analysis can be graphical or in text reports. When you choose a mode of analysis, the appropriate graph or text report appears on the Analysis Page.

The Analysis Page provides functionality for graphs or reports that appear on the page. You can:

- Change the type of graph or report that CXperf displays on the Analysis Page.
- Select Metrics to analyze.
- Select Region Types to analyze.
- Configure graphs and reports with Metric and Region options.
- Display data file information.
- Search for a region type.
- Save profiling options to a program.
- Create a second fully functional Analysis Page to compare and contrast data.
- Zoom on a graph to display a subset of the performance data.

Refer to Figure 19 on page 91 and “Toolbar” on page 92 for details.

Figure 19 Analysis Page

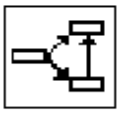


Toolbar

The following is a brief description of each toolbar option on the Analysis Page.

Summary Profile—Displays performance data in a two-dimensional graph. Data is graphed by region type.

Parallel Profile—Displays performance data in a three-dimensional graph. Data ia graphed per thread and per region type.



Call Graph—Displays a Call Graph. Performance data and routine relationships are displayed. The Call Graph metrics selection must be made during Instrumentation.

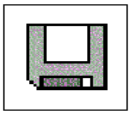
Summary Report—Displays a text report with metric information for the whole application or for an individual region type.

Parallel Report—Displays a text report with metric information for all processes and for all threads within those processes.

Call Graph Report—Displays a text report with information for caller routines and called routines. The Call Graph metrics selection must be made during Instrumentation.

Data File Information—Displays the CXperf version, the name of the executable file, the name of the PDF, the process state, metric data, and machine details for the profiling session.

Find Region—Locates a specific region type in the profiled program. Find Region is available when you have a Summary Profile, a Parallel Profile, or a Call Graph on the Analysis Page. Invokes a dialog—See Figure 20.

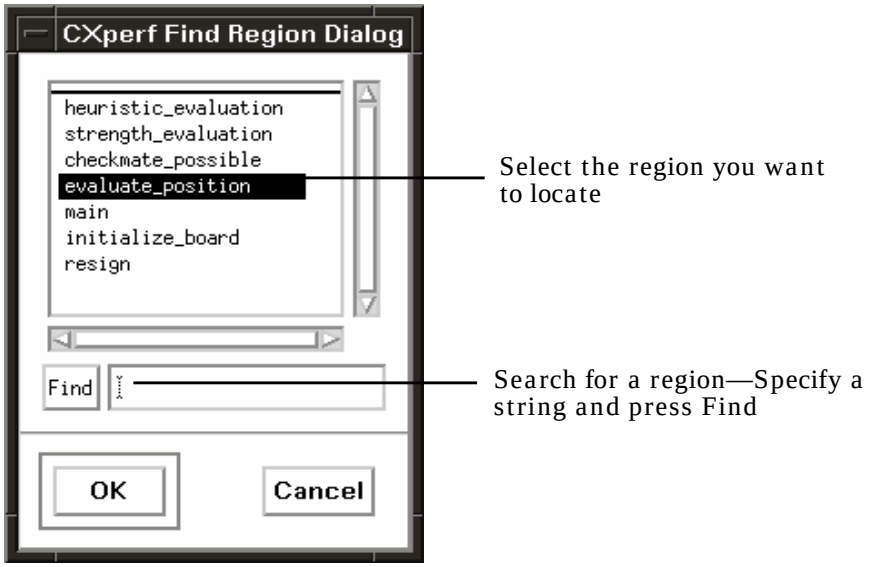


Save Profile—Saves instrumentation instructions to the program. Invokes a dialog—See Figure 21.

Tear Off Analysis—Makes a copy of the current Analysis Page by generating a second fully functional Analysis Page. Analyze several PDFs simultaneously using copies of the Analysis Page.

Find Region invokes a dialog. Figure 20 demonstrates how to use the dialog to locate region types previously selected for profiling.

Figure 20 **Find Region dialog**

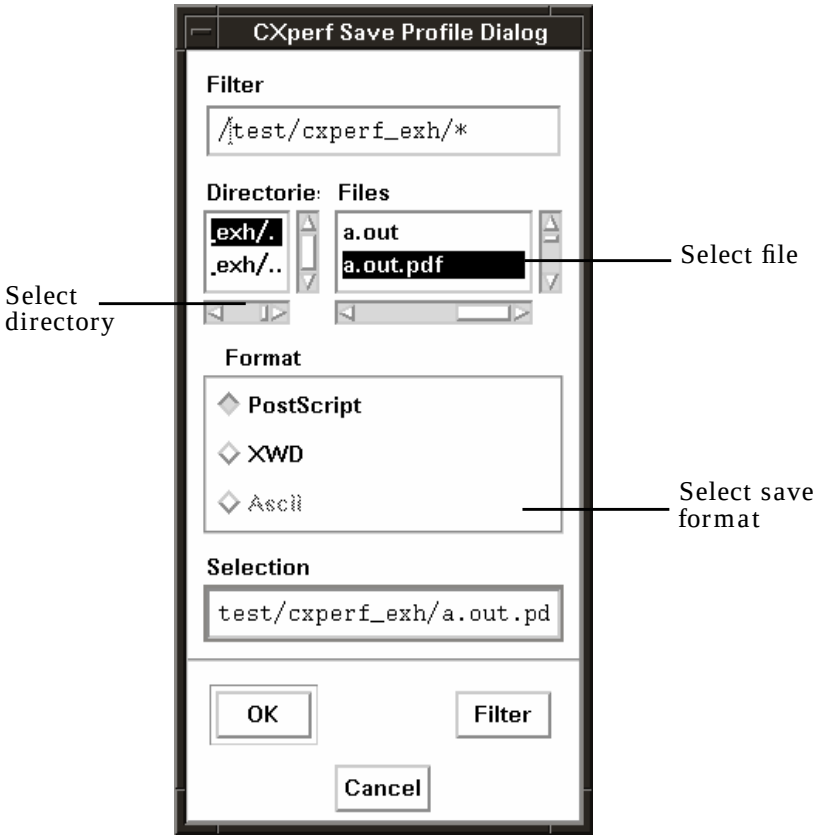


If the program contains a large number of region types, you can search for one by specifying a string in the **Find** field. The string can be a region name or part of a region name. CXperf scrolls the region type list and displays the matching name at the top of the list.

When you close the Find Region dialog by selecting **OK** the appropriate graph redraws to display the selected region type.

Save Profile invokes a dialog. Use the Save Profile dialog, as demonstrated in Figure 21, to select a format to save your file.

Figure 21 Save Profile dialog



Configuration options

Configure your PDF profiling data during analysis using the Region and Metric sections at the top of the Analysis Page.

Region

From the Region section on the Analysis Page, you can choose:

- Region Type
- Sort Criteria
- Subset Selection

Configure your Profiles or Reports using the options in the Region section, as described in Figure 22 and Table 7.

Figure 22 **Analysis Page: Region**

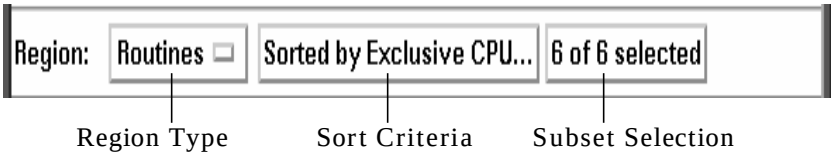


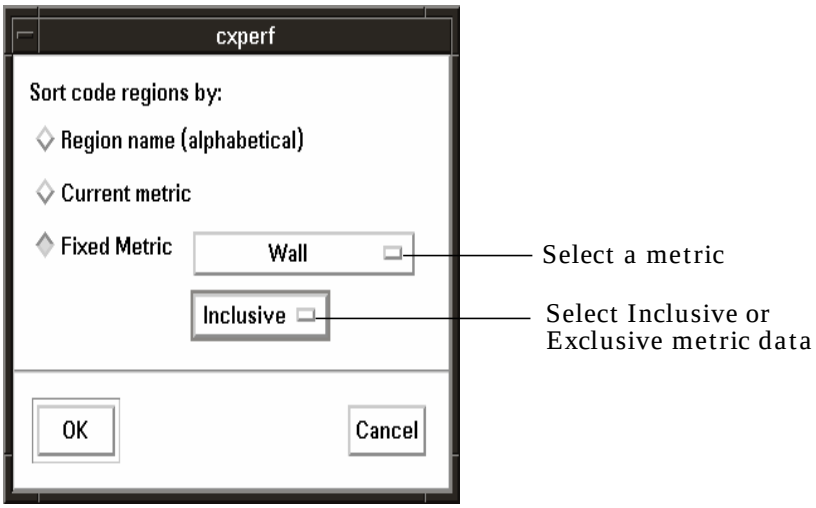
Table 7 describes the functionality for each Region button from Figure 22.

Table 7 **Region configurations**

Region selection button	Function
Region Type	Select a region type—Routines or Loops.
Sort Criteria	Specify how to sort region types. Invokes a dialog—see Figure 23. Select sorting by Region name, Current metric, or Fixed metric.
Subset Selection	Specify a subset of the profiled regions. Invokes a dialog—see Figure 24. Use the dialog to search for and select regions of interest.

The Sort Criteria button invokes a dialog, as demonstrated in Figure 23. Use it to sort the regions on your Analysis Page.

Figure 23 Sort Criteria dialog



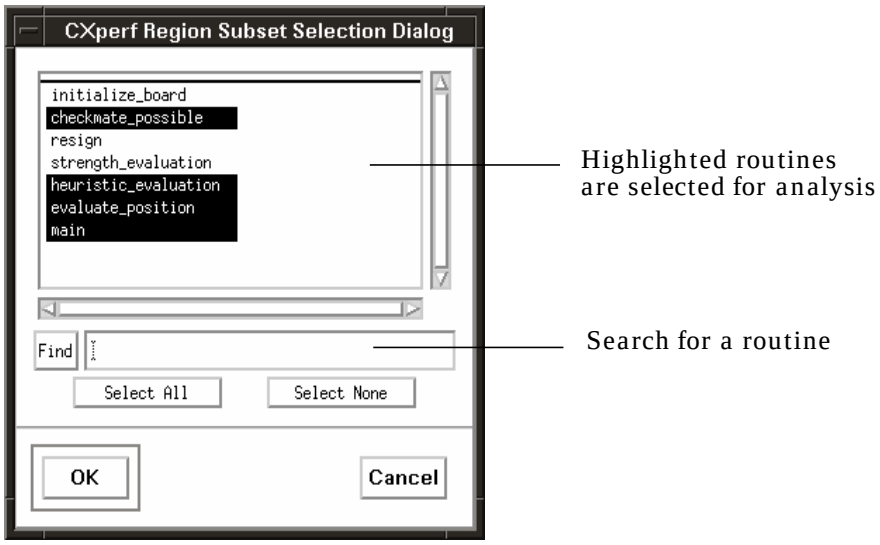
You can sort the regions on your Analysis Page according to one of the following:

Region name	Alphabetical.
Current metric	The metric currently selected on the Analysis Page Metric section.
Fixed metric	Select a metric using the option button, and choose to analyze either inclusive or exclusive metric data.

Available metrics differ according to machine architecture, and depend on the metrics you selected when you instrumented your program.

The Subset Selection button invokes a dialog, as demonstrated in Figure 24. Use it to select or deselect regions.

Figure 24 **Subset Selection dialog**



The regions selected to display on the Analysis Page are highlighted in the dialog. Select regions by highlighting them in the list.

You can search for a region by typing a string corresponding to the region name, or part of the name, in the **Find** field. The list scrolls so that the region you searched for appears near the top of the list.

Metric

From the Metric section on the Analysis Page, you can select:

- Metric
- Inclusive or Exclusive data
- Data Source

Configure your Profiles or Reports to display data of interest using the options in the Metric section as described in Figure 25 and Table 8.

Figure 25 **Analysis Page: Metric**

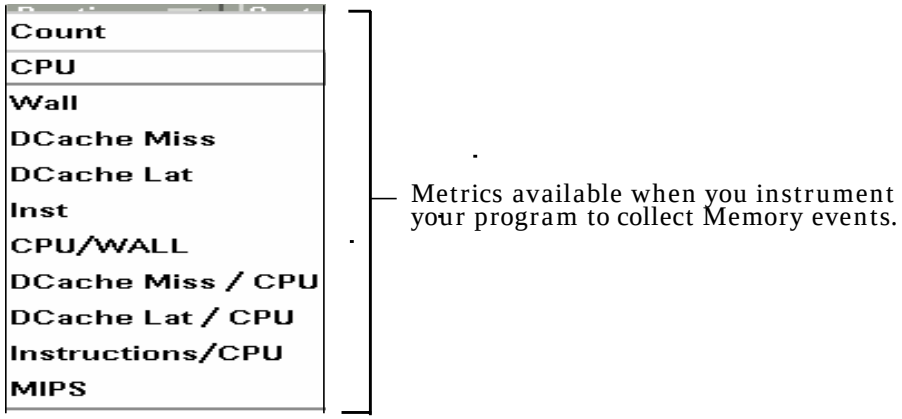
Table 8 describes the functionality for each Metric button from Figure 25.

Table 8 **Metric configurations**

Metric selection button	Function
Metric	Select a metric from those available for the current analysis—See Figure 26. The available metrics depend on the metrics you selected when you instrumented your program.
Exclusive or Inclusive	Specify whether CXperf displays metric data exclusive or inclusive of child processes.
Data Source	Specify whether CXperf displays data for the whole application, a process, or threads. Invokes a dialog—See Figure 27.

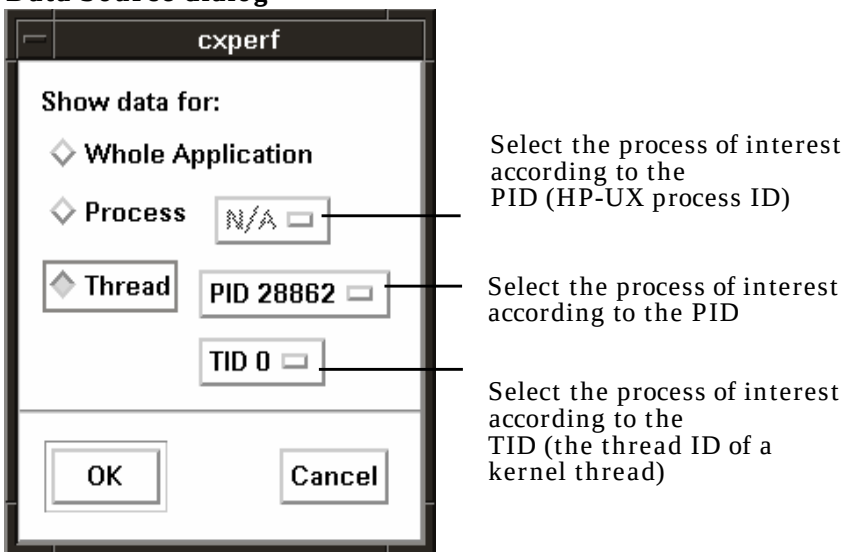
Use the Metric button to select from the metrics available. A different set of primitive and derived metrics is available, depending on your machine architecture, and how you instrumented your program. For example, Figure 26 displays metrics available when you instrument your program to collect Memory events.

Figure 26 **Select Metric**



Use the Data Source dialog to select the granularity of data display. View data for the whole application, for a single process, or for threads, as described in Figure 27.

Figure 27 **Data Source dialog**


cxperf
Show data for:
Whole Application
Process
N/A
Thread
PID 28862
TID 0
OK
Cancel
Select the process of interest according to the PID (HP-UX process ID)
Select the process of interest according to the PID
Select the process of interest according to the TID (the thread ID of a kernel thread)

Graphical Analysis

CXperf stores performance data in Performance Data Files (PDFs). Reports, both graphical and textual, are built from the PDFs.

Graphical analysis of profiling data is only available in GUI mode.

The sections below describe how to access and analyze profiling data in GUI mode.

Accessing profiling data

To access PDF information and generate graphs follow these steps:

Step 1. Open CXperf on the Analysis Page.

You can do this in one of the two following ways:

- Instrument and run your program under CXperf.

  When the program completes, the Analysis Page displays the performance data for the PDF generated during the current profiling session.

- Invoke CXperf with the name of an existing PDF using one of the following methods:

  `cxperf` *filename*`.pdf`

  `cxperf -pdf` *filename*

where

`-pdf`	Specifies the PDF to use. Use `-pdf` when the filename does not have the .pdf extension as shown in the second syntax example.
filename`.pdf`	Specifies a PDF. To use the first syntax example, the PDF name must have the .pdf extension.

filename Specifies a PDF.

Because you use the -pdf option in the second syntax example, you can use a PDF name without the .pdf extension.

CXperf starts and the Analysis Page appears.

Step 2. Use the Analysis Page Toolbar to select the type of graph:

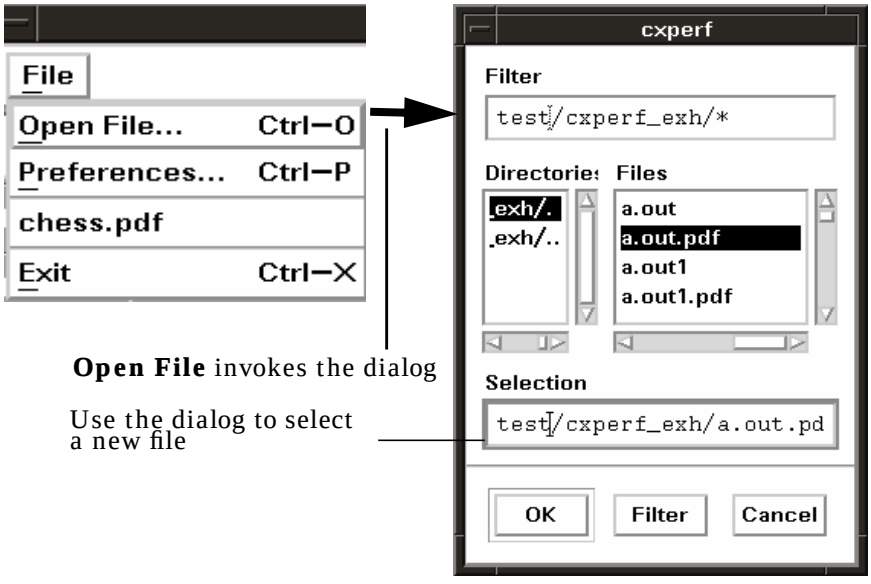
- Summary Profile
- Parallel Profile
- Call Graph

Step 3. Select the **File menu** on the Analysis Page, then **Open File** when you need to select a different PDF. Figure 28 displays the File menu and the Open File dialog that is invoked when you select **Open File**.

CXperf redraws the Analysis Page and displays the performance data for the new PDF.

Step 4. Use the Region and Metric sections on the Analysis Page to vary the configuration options for graphs and reports.

Figure 28 File menu: Open File



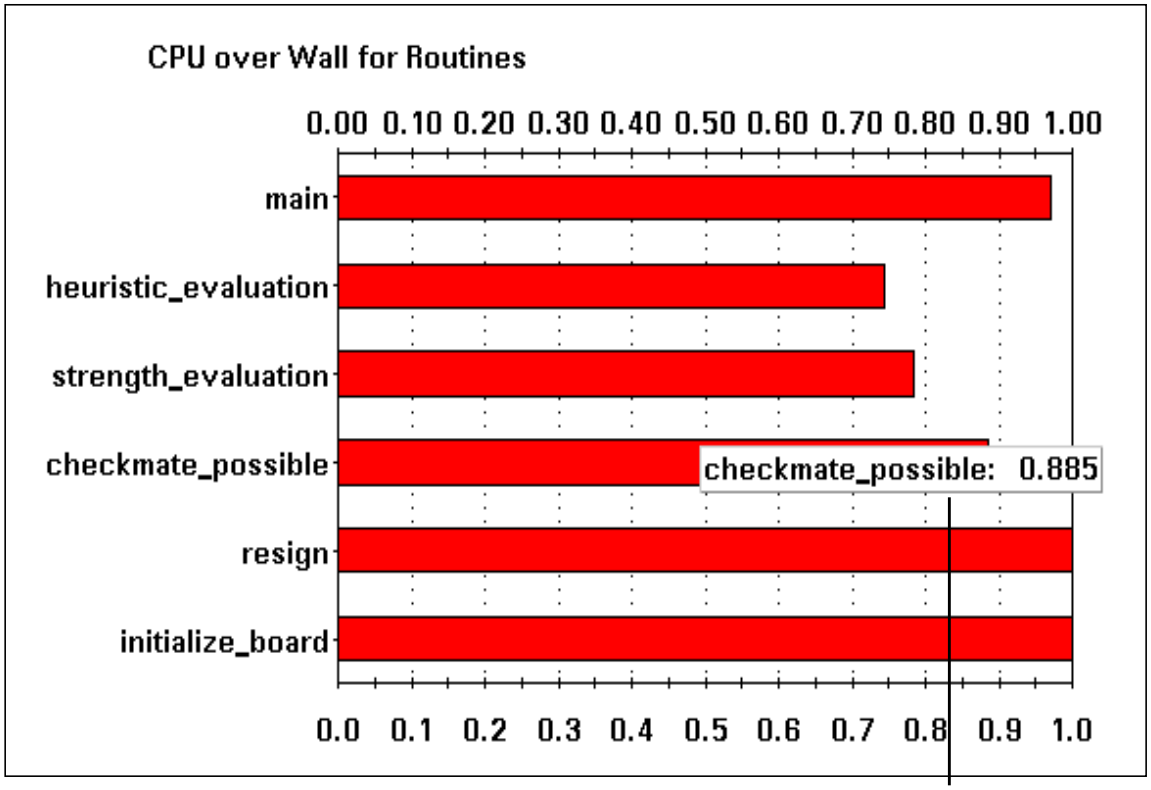
Summary Profile

The Summary Profile is a two-dimensional graph of performance data for the selected region types of your program. Figure 29 is an example of a Summary Profile displaying CPU/Wall data for six routines in a program.

The Summary Profile displays on the Analysis Page.

Use the functionality provided on the Analysis Page to vary graph configurations. Refer to “Toolbar” on page 92 and “Configuration options” on page 95 for further details.

Figure 29 Summary Profile



In the Summary Profile you can:

- Click with the left mouse button on any bar in the graph to display Region Detail associated with the corresponding region type. Figure 30 displays the Region Detail dialog.
- Use the Zoom feature when there are a large number of data items on a graph and you want to focus on a subset.
- Use the **Reset Graph** and **Show All** features to redraw the graph after you have used the Zoom feature to display a subset.
- View exact data values for a region in the graph by moving the mouse over any bar in the graph. Data values display in a pop-up window beside the mouse arrow. For example, refer to the pop-up in Figure 29 for the `checkmate_possible` routine.

Region Detail

Invoke Region Detail dialog, shown in Figure 30, by clicking with the left mouse button on any bar in the graph.

From the Region Detail dialog, you can:

- View metric values for the currently selected region.
- View a list of routines called by the currently selected routine if you selected Call Graph during Instrumentation.

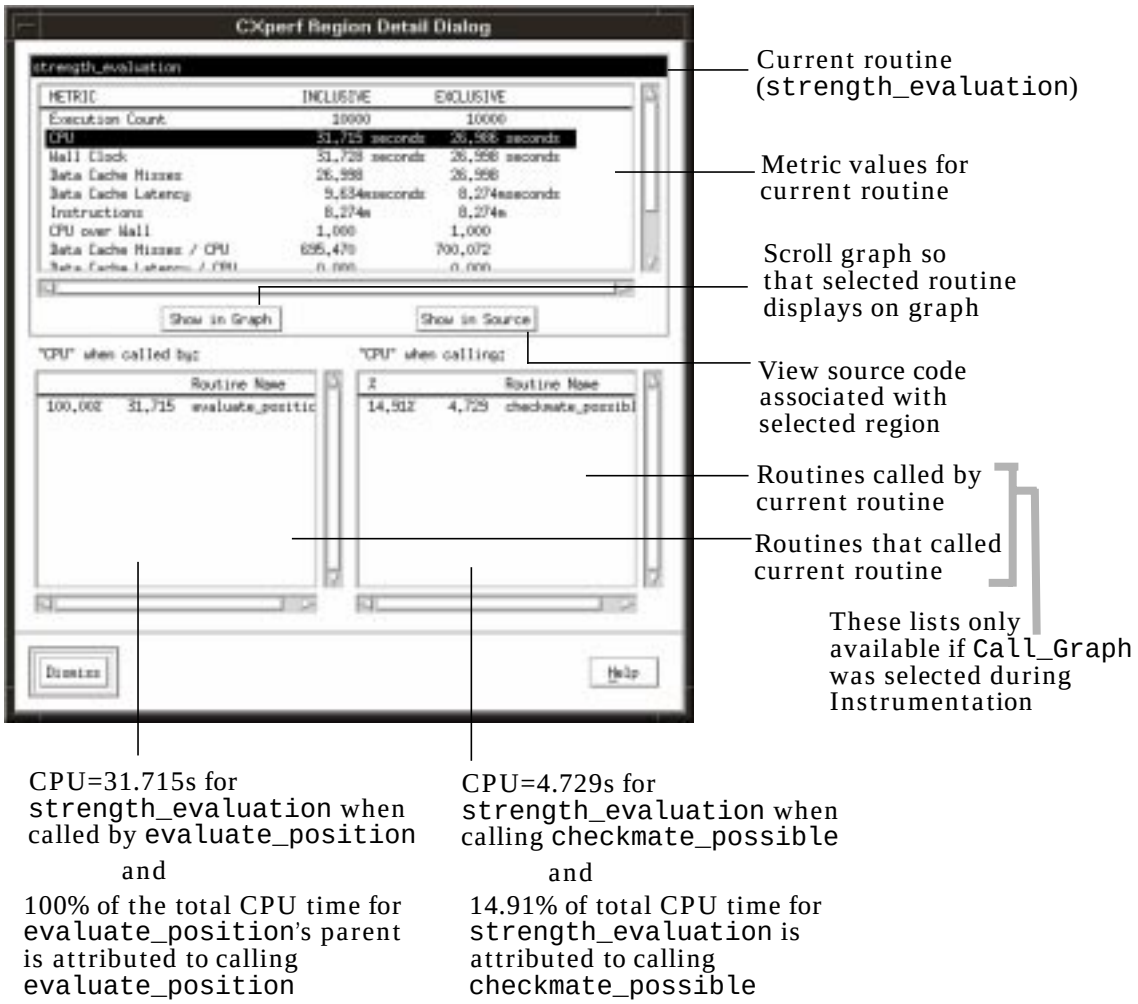
  The list is ranked by value of the current metric. The called routine that contributed the highest percentage of total metric value for the selected routine is listed first.
- View a list of routines that called the currently selected routine if you selected Call Graph during Instrumentation.

  The list is ranked by value of the current metric. The caller routine that contributed the highest percentage of total metric value for the selected routine is listed first.
- Select any region to view details for that region in the dialog.
- Select a region, and use the **Show in Graph** function to scroll the graph so that the selected region displays on the graph.

- Select a region, and use the **Show in Source** function to view source code associated with the selected region. **Show in Source** invokes a Source Window as displayed in Figure 31 on page 105.

You can invoke the Region Detail dialog by clicking with the left mouse button on any bar in a Summary Profile or a Parallel Profile. Figure 30 displays the dialog, which contains performance data for the routine you clicked, as well as source code correlation, and caller and callee information.

Figure 30 Region Detail dialog



Invoke the Source Window using **Show in Source** to view source code for the region selected in Region Detail.

Source Window

The Source Window is annotated

- To identify region types that can be profiled.

  Bold letters indicate regions currently selected for profiling. Normal letters indicate regions not selected for profiling.

  – R or r indicate routines

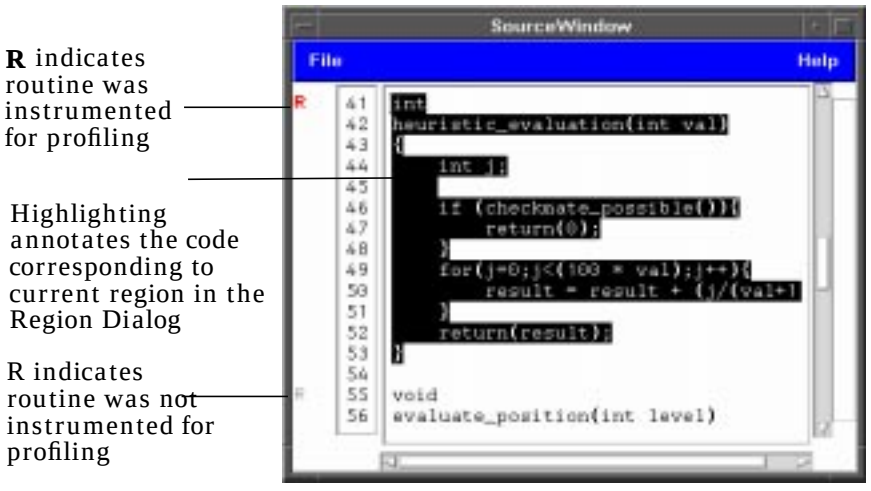
  – L or l indicate loops

  – P or p indicate parallel loops

- To identify the section of code corresponding to the selected region in the Region Detail dialog.

Figure 31 displays a Source Window annotated to indicate two routines; one is instrumented for profiling, one is not. Highlights in the Source Window annotate the code corresponding to the current region selected in the Region Detail dialog.

Figure 31 Source Window



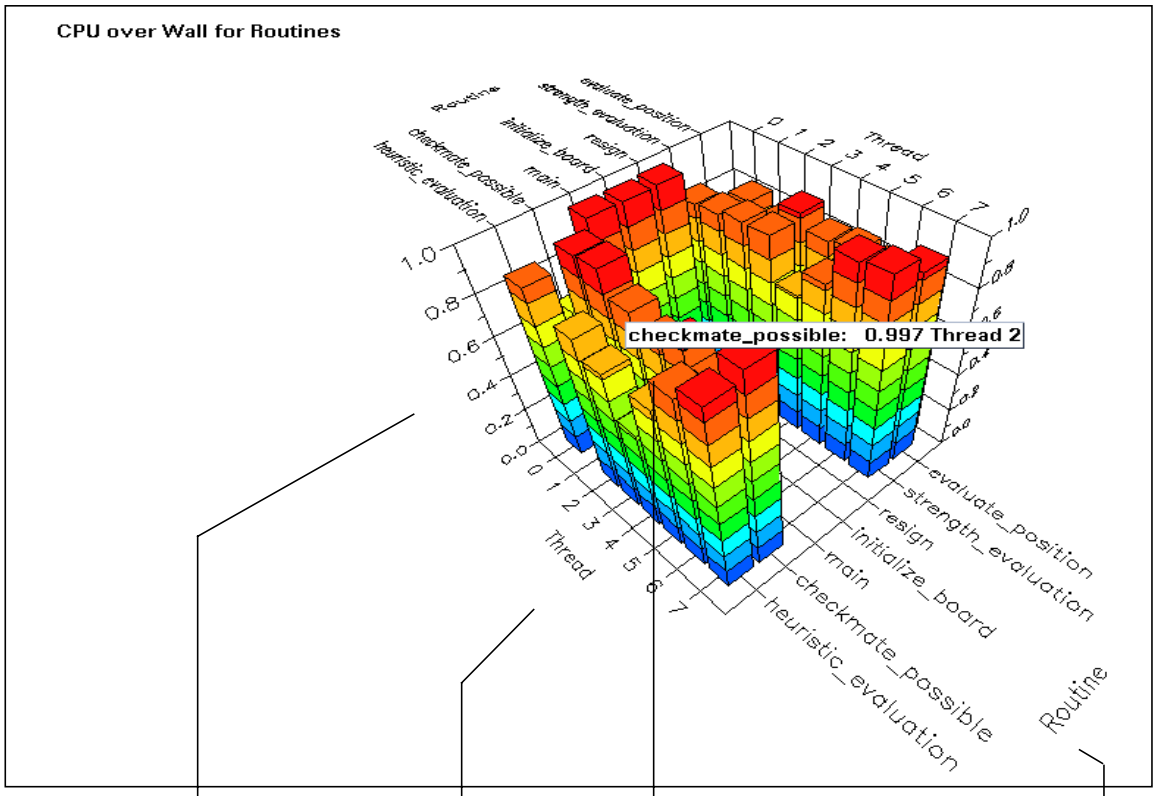
Parallel Profile

The Parallel Profile is a three-dimensional graph of performance data for the selected region types of your program. Figure 32 displays a Parallel Profile reporting CPU/Wall performance for seven routines of a program, using seven threads.

The Parallel Profile displays on the Analysis Page.

Use the functionality provided on the Analysis Page to vary graph configurations. Refer to “Toolbar” on page 92 and “Configuration options” on page 95 for further details.

Figure 32 Parallel Profile



In the Parallel Profile you can:

- Click with the left mouse button on any bar in the graph to display Region Detail associated with the corresponding region type. Refer to “Region Detail” on page 103.
- Use the Zoom feature when there are a large number of data items on a graph and you want to focus on a subset.
- Rotate the graph by placing the cursor over the graph and holding down the middle mouse button. Restrict rotation to a single axis by pressing the **x**, **y**, or **z** key while moving the mouse to rotate the graph.
- Use the **Reset Graph** button to redraw the graph to its original position after you rotate it.
- Use the **Show All** button to redraw the graph to its original position after you use the Zoom feature to display a subset.
- View exact data values for regions by moving the mouse over any bar in the graph. Data values display in a pop-up window beside the mouse arrow. For example, refer to the pop-up for the routine `checkmate_possible` in Figure 32.

Call Graph

The Call Graph displays on the Analysis Page and graphs routine relationships for selected routines in your program. Figure 33 displays an example of a Call Graph.

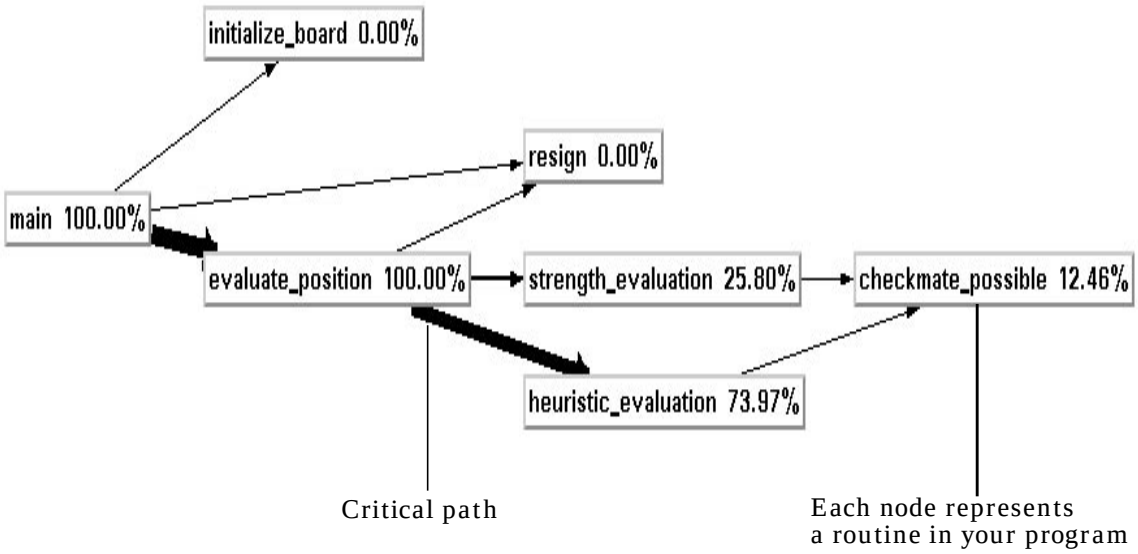
A Call Graph is available only when the Call Graph option is selected during Instrumentation.

In GUI mode, enable Call Graph metric selection in the Select metrics to collect section on the Instrumentation Page. Refer to “Selecting metrics to collect” on page 56 for details.

In line mode, specify the `call_graph` parameter of the `collect` command. Refer to “Selecting metrics to collect” on page 63 for details.

You can collect and display all metrics, except derived metrics, in the Call Graph.

Figure 33 Call Graph



The features of a Call Graph are:

- Each node of the graph represents a routine in your program.
- Arrows between the nodes point from the caller routine to the called routine.

  A thicker line between the nodes indicates the critical path of execution through the program, for the chosen metric.
- The top 10 routines, ranked by inclusive CPU, appear the first time you access a Call Graph.

  Change the metric used to rank routines using the Metric options on the Analysis Page.
- Performance data displays for a process or for individual threads.

  Change the display using the Data Source option in the Metric section of the Analysis Page.
- If there are more than 10 routines, the top 10 routines are graphed as individual nodes. The rest of the routines are collapsed into nodes. Collapsed nodes are indicated by asterisks (*) in the graph.

  Click on the astericks to expand collapsed nodes. The top *n* routines in the collapsed node—ranked by the currently selected metric—appear. The number of routines (*n*) is controlled by the **Routines Displayed** option menu at the bottom of the Analysis Page. The default is 10 routines.
- The **Recollapse** button at the bottom of the graph collapses an expanded node one level. The current depth appears to the right of the **Recollapse** button.
- The percentage of the program total for a selected metric that is attributed to each routine appears to the right of the routine name.
- Clicking with the left mouse button on any routine name in the graph displays associated Region Detail. Refer to “Region Detail” on page 103.

Refer to “Toolbar” on page 92 and “Configuration options” on page 95 for more details about changing configurations in a Call Graph.

Text Reports

CXperf stores performance data in Performance Data Files (PDFs). Reports, both textual and graphical, are built from PDFs.

Text reports are available in both GUI and line mode. Text reports are similar in each mode, but methods to access them are different.

The sections below describe how to access and analyze profiling data in GUI and line mode. Topics are:

- Accessing profiling data in GUI mode
- Accessing profiling data in line mode
- Report fields
- Summary and Parallel Reports
- Call Graph Report
- Line Mode Report

Accessing profiling data in GUI mode

To access PDF information and generate reports:

Step 1. Open CXperf on the Analysis Page.

You can do this in one of the two following ways:

- Instrument and run your program under CXperf.

  When the program completes, the Analysis Page displays the performance data for the PDF generated during the current profiling session.

- Invoke CXperf with the name of an existing PDF using one of the following methods:

  `cxperf` *filename*`.pdf`

  `cxperf -pdf` *filename*

where

`-pdf`	Specifies the PDF to use. Use `-pdf` when the filename does not have the .pdf extension as shown in the second syntax example.
filename`.pdf`	Specifies a PDF. To use the first syntax example, the PDF name must have the .pdf extension.
filename	Specifies a PDF. Because you use the -pdf option in the second syntax example, you can use a PDF name without the .pdf extension.

CXperf starts and the Analysis Page appears.

Step 2. Use the Toolbar on the Analysis Page to select the type of text report:

- Summary Report
- Parallel Report
- Call Graph Report

Step 3. Select the **File Menu** on the Analysis Page, then select **Open File** to open a different PDF. Refer to Figure 28 on page 101 for the **File** menu.

CXperf redraws the Analysis Page and displays the performance data for the new PDF.

Step 4. Use the Metric and Region sections on the Analysis Page to vary the configuration options for reports.

Accessing profiling data in line mode

Line mode commands can be incorporated into a command file or a script to perform analysis in batch mode. Refer to “Batch mode” on page 85 for details.

To access performance reports in line mode perform these steps:

Step 1. Invoke CXperf with the name of an existing PDF using one of the following methods:

`cxperf -nw` *`filename`*`.pdf`

`cxperf -nw -pdf` *`filename`*

where

`-nw`	Specifies line mode (**n**o **w**indows).
`-pdf`	Specifies the PDF to use. Use `-pdf` when the filename does not have the .pdf extension as shown in the second syntax example.
`filename``.pdf`	Specifies a PDF. To use the first syntax example, the PDF name must have the .pdf extension.
`filename`	Specifies a PDF. Because you use the -pdf option in the second syntax example, you can use a PDF name without the .pdf extension.

Step 2. Use `analyze` to display reports by entering `analyze` as shown here:

`(CXperf)` **`analyze`**

This command displays all available reports and metrics for all available regions in your program.

Using analyze

Use `analyze` to create and display text performance reports.

`analyze` generates reports from the current PDF. A PDF must exist before you can display a performance report. A PDF is created when you select region types with `select` and execute the program with `run`. If you do not specify a PDF name in the current profiling session, CXperf uses the default file name *executable*.pdf.

Refer to "Using `set pdf`" on page 115 for details about specifying a PDF name.

To analyze a PDF in line mode use the following command syntax:

`analyze` [*metric-list*] [*region-type*] [*routine-list*] [*i/o_redirection*]

where

metric_list — Specifies the metrics to display in reports. If no metrics are specified, all available metrics are displayed.

When used, this parameter should precede any other parameter. Separate multiple metrics with a space. Valid values are:

`call_graph`—Valid only if Call Graph was selected during instrumentation

`counts`—Valid for routines only

`cpu`

`wall_clock`

`events`

region_type — Specifies the region type to display in reports. Valid values are:

`routine`	Specifies routines only.
`loop`	Specifies all loops.
`pregion`	Specifies parallel loops only.
`call_graph`	Specifies Call Graph only.

You can specify only one region type. If you do not select a region type, reports display data for all profiled regions.

routine_list	Specifies one or more routines for the selected region type. Separate multiple routines with a space.
i/o_*redirection*	Redirects standard output or error to a specified file when you use one of the redirection operators (<, >, >>, >&, >>&).

CXperf displays reports in line mode using the pager specified with the PAGER environment variable. If the PAGER environment variable is not set, CXperf uses the `more` command to page output.

When you execute `analyze` without specifying any parameters CXperf displays all available reports and metrics for all profiled regions in your program.

Invoke `analyze` with the *region_type* parameter to display specific performance reports. The types of report available depend on how you prepared and instrumented the program that created the current PDF.

You can display text performance reports for the following region types:

Routines	Use `analyze routine`. Routine reports are available if you compiled the source files using `+pa` or `+pal` options or instrumented for profiling with CXoi, and selected routines during instrumentation.
Call Graphs	Use `analyze call_graph`. Call Graph reports are available if you compiled the program using `+pal` or instrumented for routine-level profiling with CXoi, and if you collected CPU and Wall Clock time.
Loops	Use `analyze loop`. Loop reports are available if you compiled the program using `+pal` at optimization level `+02` or `+03` and you selected loops during instrumentation.
Parallel loops	Use `analyze pregion`. Parallel loop reports are available if you compiled the program using `+pal` at optimization level `+03 +Oparallel` and you selected parallel loops during instrumentation.

Refer to the *CXperf Command Reference* for more details and options to configure text reports in line mode using `analyze`.

Using `set pdf`

Use `set pdf` during a CXperf session to set the name of the PDF to be created or read during the session. You may want to do this for the following reasons:

- To prevent CXperf from overwriting an existing PDF

  CXperf generates a PDF using the default name *executable*.pdf when you invoke CXperf with the name of an executable file, and run your program. If you rerun the same executable file under CXperf control, CXperf overwrites all data in the original *executable*.pdf unless you change the name of the PDF.

  Use `set pdf` to change the name of the PDF between runs of your program.

- To analyze a different PDF

  You can analyze data for several PDFs during a single profiling session. After you invoke CXperf with the name of a PDF you can analyze that PDF or change to a different one created in a previous profiling session, including PDFs created on different architectures or from different executable files.

  Use `set pdf` before `analyze` to select a new PDF during a profiling session.

To change the name of the PDF in line mode use the following syntax:

`set pdf` *filename*

where

filename	Specifies the name of a PDF. The filename you specify should have a .pdf extension.

Using `set visibility`

Use `set visibility` to set process and thread filters for analysis. By default, CXperf displays performance data in reports for each process. By setting visibility for threads you can display performance data on a thread by thread basis.

To change the visibility filter use the following syntax:

```
set visibility [ process | threads ]
```

where

`process`	Displays whole process performance data in reports.
`threads`	Displays performance data for individual threads in reports.

You can also use `set visibility` to set loop nesting levels during Instrumentation of your program. This option is discussed in “Selecting loop nesting levels” on page 61.

Refer to the *CXperf Command Reference* for more details about `set visibility`.

Using `list`

Use `list` to display lines of text from the source file for the current executable. Lines containing region types that can be selected or deselected for profiling are annotated with one or more of the following letters:

R or r	Indicate routines
L or l	Indicate loops
P or p	Indicate parallel loops

Lowercase letters indicate regions that are currently deselected, while uppercase letters indicate regions that are selected for profiling.

When you execute `list` without parameters, CXperf displays the current source file.

The syntax for `list` is as follows:

`list` [*routine* | [*filename*] [:] { *first-line* [*last-line*] | *routine*}]

where

routine	Specifies the name of a routine to display.
filename	Specifies the name of a source file to display.
first-line	Specifies a source code line number as the first line to display.
last-line	Specifies a source code line number as the first line to display.

CXperf uses the directories in its search path to find the source file. If a source file was moved after compiling, use the `add path` command to add the new directory to the CXperf search path.

Refer to the *CXperf Command Reference* for more details about `list` and `add path`.

Using `list selectable`

Use `list selectable` to display lines of source code that contain region types that can be selected for profiling. The entire source code is not displayed. Only the lines that are annotated to indicate regions available for profiling are displayed:

Lines annotated with one or more of the following letters indicate the conditions noted below:

@ or a or A	(this line is) Ambiguously referenced at one or more additional program locations
R or r	Routines
L or l	Loops
P or p	Parallel loops

Lowercase letters indicate regions that are currently deselected, while uppercase letters indicate regions that are selected for profiling.

When you execute `list selectable` without parameters, CXperf displays the lines of source code in the current source file containing selectable region types.

The syntax for `list selectable` is as follows:

`list selectable` [*routine* | [*filename*] [:] { *first-line* [*last-line*] | *routine*}]

where

routine	Specifies the name of a routine to display.
filename	Specifies the name of a source file to display.
first-line	Specifies a source code line number as the first line to display.
last-line	Specifies a source code line number as the first line to display.

CXperf uses the directories in its search path to find the source file. If a source file was moved after compiling, use the `add path` command to add the new directory to the CXperf search path.

Refer to the *CXperf Command Reference* for more details about `list selectable` and `add path`.

Report fields

This section describes column headings, abbreviations, annotations, and terms that appear in CXperf reports.

>	Symbol appears in Call Graph Reports. Indicates the primary routines in each section. Sections in the Call Graph Report are displayed in order, from highest to lowest, ranked by inclusive Wall Clock time for the section's primary routines. Routines listed above the primary routine in each section are the callers of that routine—the routine's parent. Routines listed below the primary routine in each section were called by that routine—the routine's children.
Calls in	For callers of the primary routine in each section of the Call Graph Report—the number of times the primary routine was called. For the primary routine in each section—the total number of calls made to that routine.
Calls out	For callees of the primary routine in each section of the Call Graph Report—the number of times the routine was called by the primary routine. For the primary routine in each section—the total number of times it was called.
Count	Number of times a loop executed or Execution count.
less children	Excluding values for called routines. However, if an instrumented routine calls an uninstrumented routine, CXperf is not able to separate the time spent in the uninstrumented child routine, from the time spent in the instrumented parent.
less inner	Excluding time spent in inner loops.
Line	Starting source line number for the region type—line numbers for optimized loops annotated with a lowercase letter indicate that the loop was split into two or more loops during optimization.

m	Time expressed in milliseconds. CXperf reports time in seconds except where m indicates milliseconds
N/A	Metric was not collected.
NL	Nesting level of a loop after optimization.
Optimizations (Opts)	Abbreviations for the transformations that HP compilers perform on loops. The abbreviations include:

B:*n*	Loop blocking: n is the blocking factor—the number of iterations that were blocked together.
D	Distributed.
Ds	Dynamic selection.
Hs	Hoisted.
I	Interchanged.
P	Parallel.
PS	Parallel strip mined.
pU:*n*	Loop was partially unrolled; n is the loop unrolling factor—the number of loop iterations that were unrolled.

Refer to the *Parallel Programming Guide for HP-UX Systems* for a complete discussion of optimizations performed by HP compilers.

Optimized Loops	(*cumulative, including spawn/join overhead*)—Metrics are cumulative across all threads executing in the parallel region, and include spawn and join overhead.
	(*by thread, excluding spawn/join overhead*)—Metrics are calculated on a per thread basis for all threads executing in the parallel region, and do not include spawn and join overhead.
plus children	Including values for called routines.
plus inner	Including time spent in inner loops.
Routine names	Names of all profiled routines in your program.

PS — Profiling Status. If this column is blank, the region exited normally. Other possible profiling statuses are described in Table 9.

Table 9 describes the Profiling Statuses that can appear in the PS column of your text report.

Table 9 Profiling Status

Profiling Status	Description
e	Program exited at this point.
g	Routine could not be timed due to the granularity of the clock supported on architecture.
m	Invalid time management was detected for this routine, because of an unprofitable code construct, an unprofitable command, such as `exec` or `fork`, or incorrect instrumentation in your program or library routine. To work around incorrect instrumentation, do not profile program routines or library routines that show a profiling status of m.
p	Program was paused in this routine, and timing information was incomplete.
t	Program terminated at this routine.
u	Routine could not be instrumented because it was too small to gather timing data, or contains an unrecognizable construct.
x	CPU time and associated ratios, excluding called routines and inner loops, cannot be computed accurately.
y	Wall Clock time and associated ratios, excluding called routines and inner loops, cannot be computed accurately.
. (period)	Only Call Graph Report available. The time displayed does not reflect a measured time, but a gprof-style time is inferred from available profiling data.

Summary and Parallel Reports

Summary and Parallel Reports display similar data in different configurations.

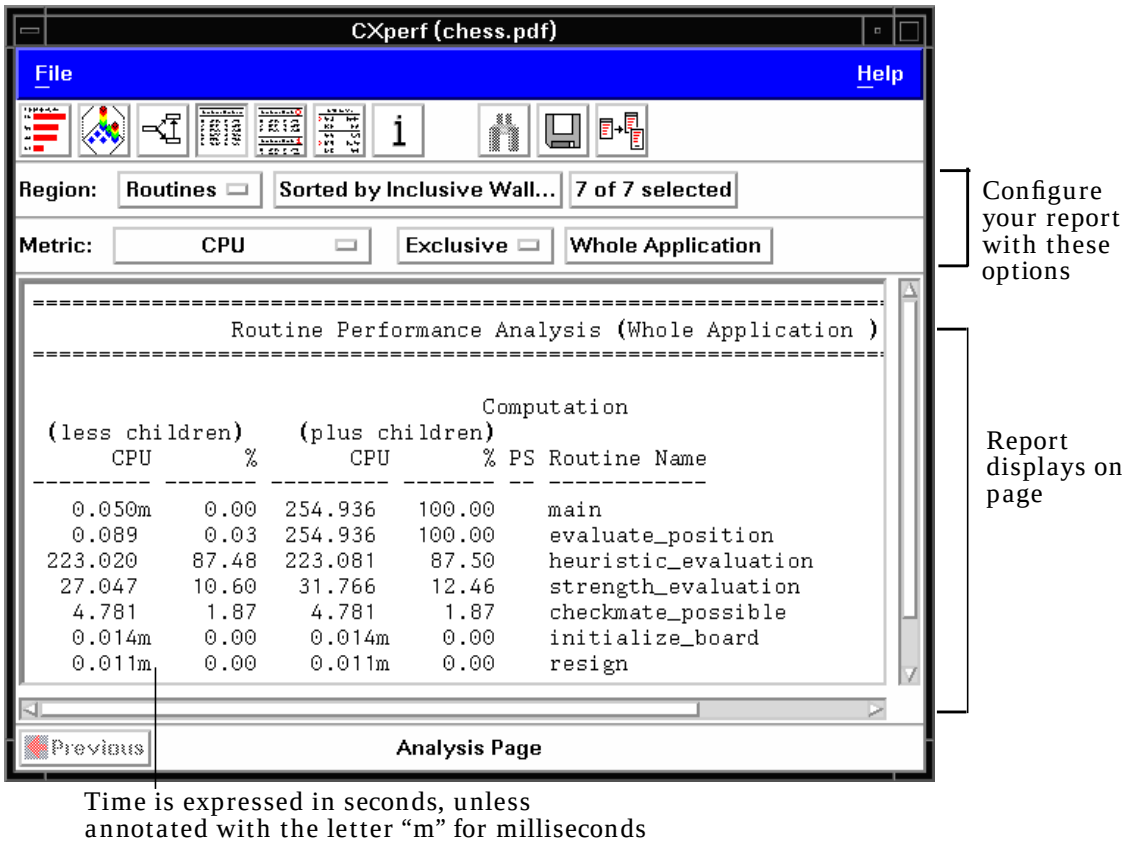
A Summary Report is typically shorter than a Parallel Report because it displays metrics for the whole application with no breakdown per thread.

A Parallel Report displays data for the whole application, broken down by all the individual processes and their threads.

For both Summary and Parallel Profiles metrics are displayed by region type—routines, loops, or parallel loops.

Figure 34 displays a Summary Report with metrics for the whole application, broken down by routines.

Figure 34 Summary Report



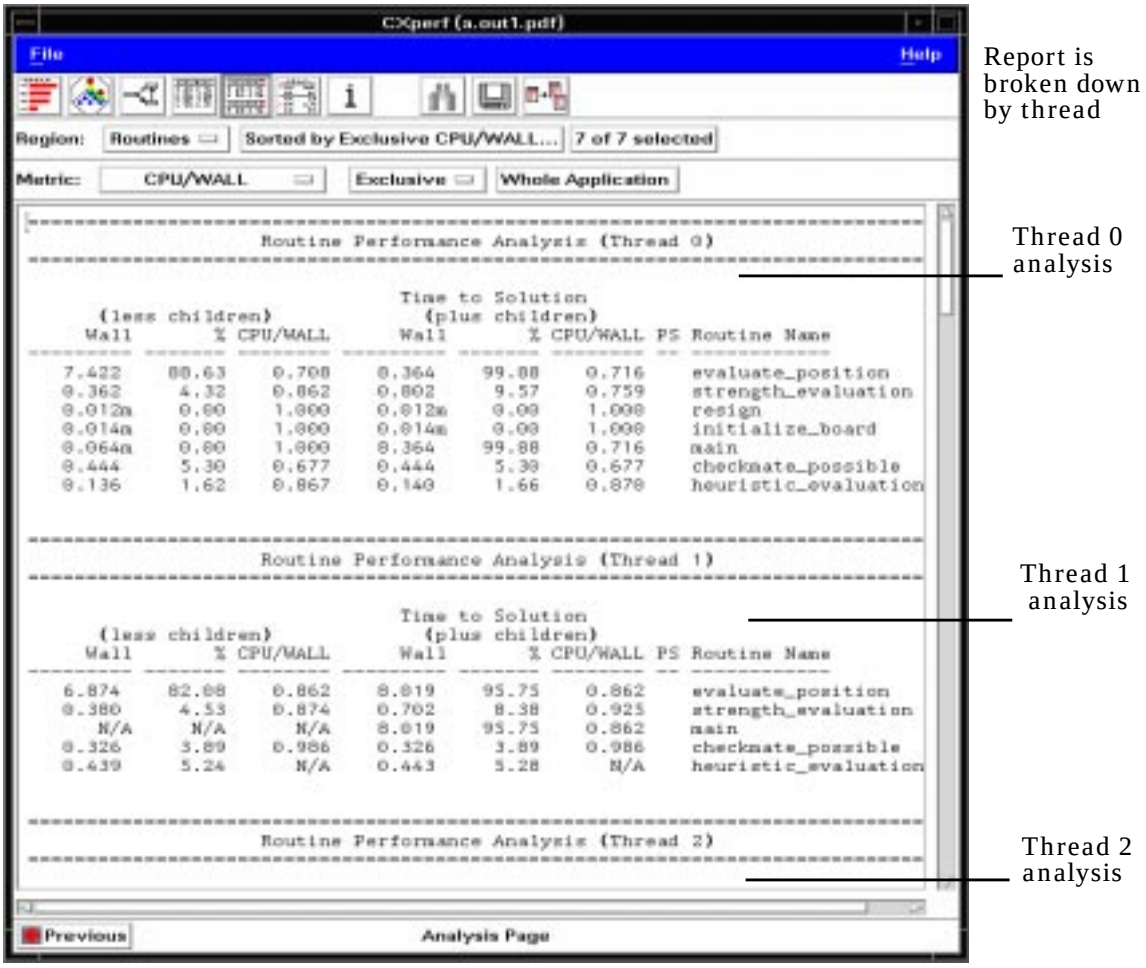
Routine Performance Analysis displays metric data inclusive and exclusive of child processes. For example, the Summary Report in Figure 34 displays typical CPU data as follows:

CPU (less children)	Total CPU time spent in each profiled routine exclusive of time spent in child processes.
% (less children)	Percentage of program's total CPU time for each profiled routine. Does not include child processes.
CPU (plus children)	Total CPU time spent in each profiled routine inclusive of time spent in child processes.
% (plus children)	Percentage of program's total CPU time for each profiled routine. Includes child processes.

By default, Summary Reports displays metrics for the whole application. You can display performance data for an individual process or thread using the Data Source button in the Metric section on the Analysis Page. The Data Source button launches a dialog where you select a single process or thread. Refer to “Metric” on page 98 and “Configuration options” on page 95 for details of the Data Source dialog and other configuration options.

Figure 35 displays a Parallel Report with metrics for the whole application, broken down for each thread within each process.

Figure 35 Parallel Report


CXperf (a.out1.pdf)
File
Help
Region: Routines
Sorted by Exclusive CPU/WALL...
7 of 7 selected
Metric: CPU/WALL
Exclusive
Whole Application
Routine Performance Analysis (Thread 0)
Time to Solution
(less children)
(plus children)
Wall % CPU/WALL Wall % CPU/WALL PS Routine Name
7.422 88.63 0.708 8.364 99.88 0.716 evaluate_position
0.362 4.32 0.862 0.802 9.57 0.759 strength_evaluation
0.012m 0.00 1.000 0.012m 0.00 1.000 resign
0.014m 0.00 1.000 0.014m 0.00 1.000 initialize_board
0.064m 0.00 1.000 8.364 99.88 0.716 main
0.444 5.30 0.677 0.444 5.30 0.677 checkmate_possible
0.136 1.62 0.867 0.140 1.66 0.870 heuristic_evaluation
Routine Performance Analysis (Thread 1)
Time to Solution
(less children)
(plus children)
Wall % CPU/WALL Wall % CPU/WALL PS Routine Name
6.874 82.08 0.862 8.019 95.75 0.862 evaluate_position
0.380 4.53 0.874 0.702 8.38 0.925 strength_evaluation
N/A N/A N/A 8.019 95.75 0.862 main
0.326 3.89 0.986 0.326 3.89 0.986 checkmate_possible
0.439 5.24 N/A 0.443 5.28 N/A heuristic_evaluation
Routine Performance Analysis (Thread 2)
Previous
Analysis Page
Report is broken down by thread
Thread 0 analysis
Thread 1 analysis
Thread 2 analysis

By default, Parallel Reports display metrics for the whole application, for each process and thread. You can display performance data for individual processes or for individual threads using the Data Source button in the Metric section on the Analysis Page. The Data Source button launches a dialog where you select process or thread. Refer to “Metric” on page 98 and “Configuration options” on page 95 for details of the Data Source dialog and other configuration options.

Metrics in Summary and Parallel Reports are displayed in Routine, Loop, and Parallel Loop Performance Analysis sections. The sections available for a given report depend on your program and the selections you made during instrumentation.

The following sections describe details for the different Performance Analyses in Summary and Parallel Reports.

Routine Performance Analysis

Routine Performance Analysis is available if:

- You compiled the program with the `+pa` option
- You selected routines during instrumentation

Use the Routine Performance Analysis section of a report to examine:

- Total time spent in each profiled routine
- Percentage of program's total time for each profiled routine, broken down by thread
- Metric value attributed to each profiled routine

Routine Performance Analysis displays metric data inclusive and exclusive of child processes. For example, the Summary Report in Figure 34 displays typical CPU data.

Loop Performance Analysis

Loop Performance Analysis is available if:

- Your program contains loops and you selected them for profiling.
- You compiled your program with the `+pal` option.
- Routines containing the loops were compiled at optimization level `+02` or `+03`.
- At least one profiled loop executed.

The loop nesting level setting affects the number of loops selected for profiling. The default loop nesting level setting selects only loops at nesting level 0—outermost loops—for profiling.

Refer to "Selecting loop nesting levels" on page 53 to select nesting levels in GUI mode, and "Selecting loop nesting levels" on page 61 to select nesting levels in line mode.

Loop Performance Analysis displays metric data inclusive and exclusive of child processes. Loop performance information also includes a history of the optimizing transformations the compiler performed on the loops. The transformations are described as abbreviations in the report, under the Opts column. The abbreviations include:

B:*n*	Loop blocking: n is the blocking factor—the number of iterations that were blocked together.
D	Distributed.
Ds	Dynamic selection.
Hs	Hoisted.
I	Interchanged.
P	Parallel.
PS	Parallel strip mined.
pU:*n*	Loop was partially unrolled; n is the loop unrolling factor—the number of loop iterations that were unrolled.

Analysis of the optimizations and metric data provides a good picture of loop performance for your program. For example, a low CPU/Wall ratio indicates performance bottlenecks caused by one or more of the following:

- I/O calls—`read()` or `write()`
- System calls—`open()` or `close()`
- Memory access misses—cache misses

For serial loops, a high CPU/Wall Clock ratio, approaching 1.0, indicates regions are compute bound.

Parallel Loop Performance Analysis

Parallel Loop Reports are available if:

- Routines were compiled at optimization level `+O3` and `+Oparallel`.
- At least one profiled loop was executed.

Parallel Loops are annotated by a **P** in the Optimization column.

A Parallel Loop Performance Analysis contains two sections:

- Optimized Loops (cumulative, including spawn/join overhead)

  The metrics are cumulative across all threads executing in the parallel region, and include spawn and join overhead. These values are graphed in the Summary Profile for parallel loops.

- Optimized Loops (by thread, excluding spawn/join overhead)

  The metrics are calculated on a per thread basis for all threads executing in the parallel region, and do not include spawn and join overhead. These values are graphed in the Parallel Profile for parallel loops.

For parallel loops the CPU/Wall Clock ratio is the concurrency factor. Values of CPU/Wall Clock time approaching *n*, where *n* is the number of processors used, indicate good parallel concurrency.

For example, as you increase the number of processes or the amount of work, the data set size, the concurrency factor should increase proportionately. This indicates the parallel loop region is scaling well in parallel.

Call Graph Report

Call Graph Reports display:

- Inclusive and exclusive metric data for each profiled routine in your program

  All metrics, except derived metrics, can be collected and displayed in the Call Graph.

- The relationships between routines—which routines are callers and which are callees

A Call Graph is available only when the Call Graph option is selected during Instrumentation.

- In GUI mode, enable Call Graph metric selection in the Select metrics to collect section on the Instrumentation Page.

  Refer to “Selecting metrics to collect” on page 56 for details.

- In line mode, specify the `call_graph` parameter for the `collect` command.

  Refer to “Selecting metrics to collect” on page 63 for details.

Figure 36 displays a sample Call Graph Report with several sections. Sections are displayed in order, from highest to lowest, ranked by inclusive Wall Clock time for the section's primary routines.

Figure 36 Call Graph Report

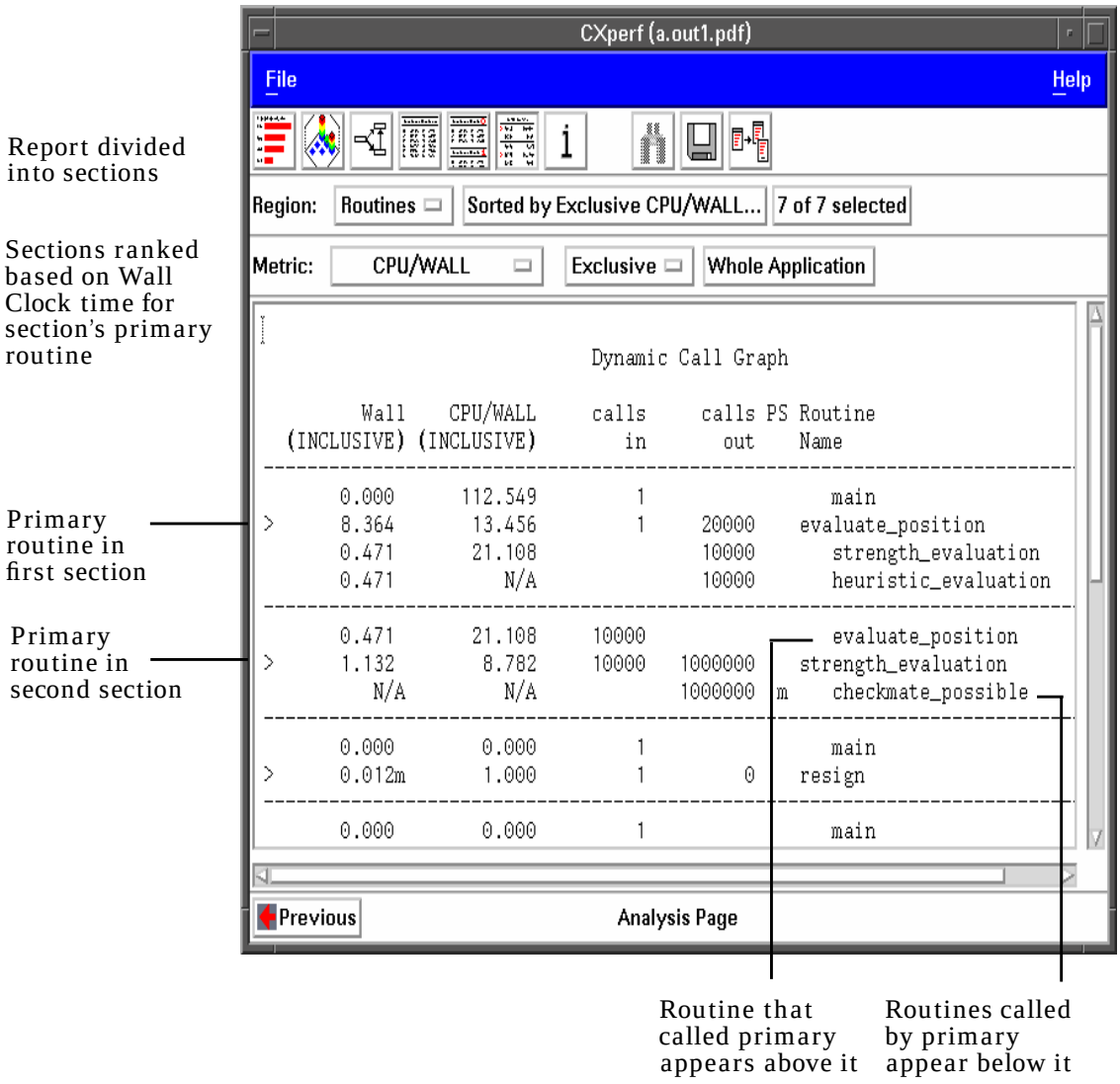


Figure 36 displays a sample report with several sections. The following list describes the interpretation of the first section:

- The > symbol indicates `evaluate_position` is the main routine of the program—has the largest Wall Clock time.
- The `Calls out` column for the main routine indicates that `evaluate_position` made 2000 calls—`1000 calls to strength_evaluation` and 1000 calls to `heuristic_evaluation`.
- The `Wall(Inclusive)` column indicates that an equal amount of Wall Clock time is spent in each of the routines called by `evaluate_position`.
- The `Calls in` column indicates that `evaluate_position` was called once. The routine listed above the primary routine in the report is the caller routine—in this example, `main`.
- The `Calls in` column in `main` indicates the number of times `main` called the primary routine.
- The `m` in the Profiling status (PS) column indicates that CXperf was unable to collect some timing information for `heuristic_evaluation.`
- The `N/A` in the CPU column for `heuristic_evaluation` indicates that this metric was not collected.

For more details about report interpretation, refer to “Report fields” on page 119.

Line Mode Report

Line Mode Reports display profiling data as specified by the `analyze` command.

CXperf displays reports in line mode using the pager specified with the PAGER environment variable. If the PAGER environment variable is not set, CXperf uses the `more` command to page output.

Using `analyze`

Use `analyze` to specify the type of report you want to display.

Refer to "Using `analyze`" on page 113 and the *CXperf Command Reference* for details and options to configure text reports in line mode.

For example, the following command sends the results of `analyze` to a file named textreport:

```
(CXperf) analyze > textreport
```

Figure 37 displays an example of a Line Mode Report.

Using `set pdf` and `set visibility`

Use `set pdf` before `analyze` to select a new PDF during a profiling session. You can analyze data for several PDFs during a single profiling session. After you invoke CXperf with the name of a PDF you can analyze that PDF or change to a different one created in a previous profiling session, including PDFs created on different architectures or from different executable files. Refer to "Using `set pdf`" on page 115 and the *CXperf Command Reference* for details.

Use `set visibility` to set process and thread filters for analysis. By default, CXperf displays performance data in reports for each process. By setting visibility for threads you can display performance data on a thread by thread basis. Refer to "Using `set visibility`" on page 116 and the *CXperf Command Reference* for details.

Figure 37 is an example of a Line Mode Report. CXperf uses `more` to page output for this example.

Figure 37 Line Mode Report

```
================================================================================
CXperf Version 6.0  Profile

Executable      : /tmp_mnt/test/cxperf_exh/a.out1
Profile Data    : /tmp_mnt/test/cxperf_exh/a.out1.pdf
Process State   : exited
CPU Time        :  117.410
Wall Clock Time :    8.374
Architecture    : HP9000/800 (16 threads)
================================================================================

================================================================================
               Routine Performance Analysis (Whole Application )
================================================================================

                                  Call Counts
    Count PS Routine Name
--------- -- ------------
  1010000    checkmate_possible
    10000    strength_evaluation
    10000    heuristic_evaluation
        1    initialize_board
        1    resign
        1    evaluate_position
        1    main

                                  Computation
 (less children)    (plus children)
     CPU       %       CPU       % PS Routine Name
-------- ------- -------- ------- -- ------------
  75.185    64.03  112.549    95.86    evaluate_position
   5.089     4.33    9.941     8.46    strength_evaluation
  0.064m     0.00  112.549    95.86    main
  0.014m     0.00   0.014m     0.00    initialize_board
  0.012m     0.00   0.012m     0.00    resign
     N/A      N/A      N/A      N/A  m checkmate_possible
     N/A      N/A      N/A      N/A  m heuristic_evaluation

(m) Missing Time: Unable to Collect or Exceeds Reasonable Limits

                               Time to Solution
     (less children)              (plus children)
    Wall       % CPU/WALL      Wall       % CPU/WALL PS Routine Name
-------- ------- -------- --------- ------- -------- -- ------------
   7.422    88.63   10.129     8.364    99.88   13.456    evaluate_position
   5.866    70.04      N/A     5.870    70.09      N/A    heuristic_evaluation
  :
  :
```

In the example above, because no parameters were specified for `analyze`, CXperf displays all available reports and metrics, for all profiled regions in the program.

The fields in a Line Mode Report and their interpretation are similar to those for Summary and Parallel Reports. Refer to “Report fields” on page 119 for details.

Viewing source in line mode

You can use `list` or `list selectable` to display lines of text from the source files that were compiled to form the current executable.

When you use `list`, CXperf displays the source code for the program with annotations at the regions available for profiling.

When you use `list selectable`, the entire source code is not displayed—only the lines that are annotated to indicate the regions available for profiling are displayed.

Lines annotated with one or more of the following letters indicate the conditions noted below:

@ or a or A	(this line is) Ambiguously referenced at one or more additional program locations
R or r	Routines
L or l	Loops
P or p	Parallel loops

Lowercase letters indicate regions that are currently deselected, while uppercase letters indicate regions that are selected for profiling.

Glossary

cache A small high-speed buffer memory used to hold those portions of the contents of the memory, that are, or are believed to be, in current use. Cache memory is physically separate from main memory, and can be assessed with substantially less latency.

clone A compiler-generated copy of a loop or a procedure. When the HP compilers generate code for a parallelizable loop, they generate two versions: a serial clone and a parallel clone. See also dynamic selection.

coherency A term applied to caches. Cache coherency is the state that is achieved when multiple processors' caches, on a multiprocessor system, always have the latest value for a data item. If a data item is referenced by a particular processor on a multiprocessor system, the data is copied into that processor's cache, and is updated there. If a second processor references the data while a copy is still in the first processor's cache, the cache coherency mechanism is needed to ensure that the second processor does not use the outdated copy of the data from memory.

Call Graph Wall clock and CPU time (inclusive and exclusive of child processes), Call counts, and metrics for each profiled routine, its parents, and its children.

concurrent In parallel processing, threads that can execute at the same time are called concurrent threads.

compiler A computer program that translates computer code written in a high-level programming language, such as C, into an equivalent machine language.

Context Switch Occurs when a process changes its state. The possible states for a process are *running*, *ready*, or *waiting/blocked*. Can be voluntary or involuntary (forced).

CPU Time Time the processors work on the process, not including time waiting for I/O or running other programs. If a process can run multiple processors the CPU time may be greater than the Wall Clock time.

CPU/Wall Clock Ratio of CPU to Wall Clock time. This is a derived metric, computed by CXperf during analysis of profiling data.

Data TLB miss Data Translation Lookaside Buffer (DTLB) miss. Represents the number of times the address translation from virtual to physical memory for data to be referenced was not found in the TLB.

dynamic selection The process by which the compiler chooses the appropriate runtime clone of the loop. See also clone.

exclusive Exclusive times and metrics reported by CXperf do not include time spent in or metrics collected for called, or child, routines.

explicit parallelism Programming style that requires you to specify parallel constructs directly. Using the MPI library is an example of explicit parallelism.

granularity Measure of the work done between synchronization points. Fine-grained applications focus on execution at the instruction level of a program. Such applications are load balanced but suffer from a low computation/communication ratio. Coarse-grained applications focus on execution at the program level where multiple programs may be executed in parallel.

hoist An optimization process that moves a memory load operation from within a loop to the basic block preceding a loop.

inclusive Inclusive times and metrics reported by CXperf include time spent in or metrics collected for called, or child, routines.

inlining A compiler optimization where selected function calls are substituted with copies of the function's object code. Inlining may result in larger executable files and greater compilation time.

Instruction counts Number of completed instructions.

Instruction TLB miss Instruction Translation Lookaside Buffer (ITLB) miss. Represents the number of times the address translation from virtual to physical memory for an instruction was not found in the TLB.

interchange Loop interchange—the reordering of nested loops. Loop interchange is generally done to increase the granularity of the parallelizable loops, or to allow more efficient access to loop data.

Latency Amount of time spent accessing memory to locate data or instructions not found in the processor's data or instruction cache.

loop blocking A loop transformation that strip mines and interchanges a loop to provide optimal reuse of encached loop data.

loop interchange The reordering of nested loops. Loop interchange is generally done to increase the granularity of the parallelizable loops, or to allow more efficient access to loop data.

Migration Occurs after a context switch when a process changes the CPU on which it runs.

MPI (Message Passing Interface) A message passing and process control library. For information on the Hewlett-Packard implementation of MPI, refer to the *HP MPI User's Guid*e.

MIPS Millions of instructions per second. CXperf calculates MIPS during analysis if Instruction counts, clock cycles, and Wall Clock time are collected.

Optimization The refining of application software programs to minimize processing time. Optimizations take maximum advantage of a computer's hardware features and minimizes input/output traffic and processor idle time.

Optimization level The degree to which source code is optimized by the compiler. The HP Fortran 77, Fortran 90, ANSI C, and ANSI C++ compilers have six levels of optimization:`+O0`, `+O1`, `+O2`, `+O3`, `+O4`, and `+Oparallel`.

Page Fault Occurs when a process requests data not currently in memory, requiring the operating system to retrieve the page containing the requested data from disk.

PID HP-UX process identification number.

PVM (Parallel Virtual Machine) A message passing and process control library.

RISC Reduced Instruction Set Computer. An architectural concept that applies to the definition of the instruction set of a processor. A RISC instruction set is an orthogonal instruction set that is easy to decode in hardware, and for which a compiler can generate highly optimized code.

strip mining The transformation of a single loop into two nested loops. Conceptually, this is how parallel loops are created.

thread An independent execution stream by a CPU. One or more threads, each of which can execute on a different CPU, make up a process. Memory, files, signals, and other process attributes are generally shared among threads in a given process. Threads are created and terminated by instructions that can be automatically generated by HP parallel compilers, inserted by adding compiler directives to source code, or coded explicitly using library calls or assembly language.

TID Kernel thread identification number for each thread executing a parallel region.

TLB Translation Lookaside Buffer (see description for definition).

Translation Lookaside Buffer A cache of virtual-to-physical memory address translations for the most recently referenced page table entries.

Wall Clock Time to solution for a process, including process idle time.

Index

Symbols

+O0, 33
+O1, 33
+O2, 12, 33
+O3, 12, 33
+O4, 33
+Oall, 33
+Onoinline, 34, 35
+Oparallel, 12
+Oprocelim, 33
+pa, 12, 33
+pal, 12, 33

Numerics

2D graph *See* Summary Profile
3D graph *See* Parallel Profile

A

Abort button, 19
accessing profiling data
  GUI mode, 100–109
  line mode, 112
All/None button, 15
Analysis Page, 20, 90
  Call Graph, 108
  Call Graph Report, 128
  configuration options, 95–99
  Parallel Profile, 106
  Parallel Report, 122–127
  performance report fields, 119
  Summary Profile, 102
  Summary Report, 122–127
  Text Reports, 110
  Toolbar, 92
analyze command, 70, 112
analyzing
  Analysis Page, 90
  analyze command, 70
  graphical reports, 100–109
  GUI mode, 20, 90–109, 122–130
    accessing data, 110
    Region Type, 95
    Show in Graph button, 103
    Show in Source button, 103
    Sort Criteria, 95
    Subset Selection, 95
    text reports, 110
    Toolbar options, 92
  line mode, 28, 112–121, 130–133
    accessing data, 112
    analyze command, 113
    list command, 116
    list selectable command, 117
    set pdf command, 115
    set visibility command, 116
    text reports, 110
  merged PDFs, 81
  report fields, 119
  selecting a PDF, 101, 112
  uninstrumented routines, 78
  viewing source
    GUI mode, 103–104
    line mode, 133
architecture-dependent metrics, 44
architecture-independent metrics, 43
archive libraries
  instrumenting and linking, 36
  instrumenting with CXoi, 35, 38
assembler, specify path name, 38

B

batch mode, 6, 85

C

cache, 135
cache coherency, 135
Cache misses, 45
Call Graph, 43, 92, 108
  critical path, 8, 108
  effect of uninstrumented routines in, 78
  nodes, 108, 109
  overview, 8
  ranking routines, 109
  Recollapse button, 109
  Routines Displayed option, 109
  selecting in GUI mode, 56, 108
  selecting in line mode, 63, 108
Call Graph Report, 92, 128
  effect of uninstrumented routines in, 78
called routines, 104, 119
caller routines, 104, 119
Calls in, 119
Calls out, 119
child processes (routines), 77, 104
clone, 135
coherency, 135
collect command, 25, 28, 63
command files, 85
command line
  batch mode, 85
  options
    -c, 35
    -e, 86
    -nw, 24, 58
    -o, 38
    -tm, 66
    -x, 85
  shell scripts, incorporating, 86
  shortcuts, 28
Compilation Page, 32, 49
  See also compiler instructions
compiler instructions, 23
  See also Compilation Page
compiler optimizations
  +O2, 12
  +O3, 12
  +O4, 33
  +Oall, 33
  +Onoinline, 34

+Oparallel, 12
+Oprocelim, 33
compiler-generated loops, 51, 58, 114
compilers, 135
ANSI C, 14
ANSI C++, 14
Fortran77, 14
Fortran90, 14
PA-RISC targeting, 38
compiling, 14, 32
instructions, 23
object files and archive libraries, 35, 38
syntax, 34
using CXoi, 35, 38
compiling and linking
in one step, 35
separately, 35
compute-bound regions, 126
concurrency, 127, 135
configuration options
reports in GUI mode, 95–99, 122
reports in line mode, 112–119
Context switch, 44, 135
Continue button, 19
Counts, 119
CPU time, 43, 135
CPU/Wall, 43, 135
critical path, 8, 108
CXmerge, 80
command syntax, 80
cxmerge command, 80
CXoi, 35, 38
command syntax, 38
instrumenting with, 39
limitations, 40
linking instrumented files, 39
cxoi command, 38
CXperf
command line shortcuts, 28
compiler options, 12, 33
graphical analysis, 90–109
interfaces, 5
product overview, 3
profiling session overview, 12
starting, 4
in batch mode, 85
in GUI mode, 14
in line mode, 24
with a PDF, 83
text reports
GUI mode, 110, 122–130
line mode, 112–121, 131–133
CXPERF environment variable, 24
cxperfmon.o, 33

D

Data and Instruction TLB misses
Data TLB misses, 46
instruction counts, 46, 136
Instruction TLB misses, 46, 136
Data Cache Utilization
Cache misses, 45
Instruction TLB misses, 45, 136
Latency, 45, 136
data collection routines
See cxperfmon.o
Data File Information, 92
data sampling points, 74
Data Source button, 98
Data Source dialog, 99
Data TLB misses, 45, 136
defaults
instrumentation settings, 51
loop nesting levels, 54, 61
metrics, 43
PDF names, 68
regions selected for profiling
GUI mode, 51
line mode, 58
derived metrics, 46
deselect command, 24, 58
dynamic selection, 136

E

environment variables
CXPERF, 24
PAGER, 28, 114, 131
PROFDIR, 66, 80
TMPDIR, 40
event-based sampling, 2
exclusive data, 98, 119, 136
Exclusive/Inclusive button, 98
executing
GUI mode, 19
pausing program, 19
process states, 19
line mode, 27
using command files, 85
using shell scripts
Execution Counts, 43, 119
Execution Page, 19
explicit parallelism, 136

F

File Menu, 101
Find Region, 92, 93
fixed loop nesting, 54, 61

G

glossary, 135
gprof, 2
granularity, 136
graphical analysis, 100–109
overview, 7
Reset Graph button, 103, 107
Show All button, 103, 107
Zoom feature, 103, 107
graphs, 91, 102, 106, 108
GUI mode
accessing profiling data, 110
Analysis Page, 91

Analysis Page Toolbar, 92
analyzing reports
graphical, 90–109
textual, 122–130
compiling, 14, 32
executing, 19
instrumenting, 49–58
interface, 5
invoking CXperf with a PDF, 83

H

hoist, 136
See also Optimizations

I

inclusive, 120
inclusive data, 98, 136
inlining, 35, 136
Instruction counts, 45, 46, 136
Instruction TLB misses, 45, 136
Instrumentation Page, 50
Call Graph, 56
loop nesting level, 53
metrics, 56
Preinstrument Executable, 57, 67
regions, 52
instrumenting
default settings, 51
in GUI mode, 15
fixed loop nesting, 54
loop nesting, 53
metrics, 56
relative loop nesting, 55
routines and loops, 49
in line mode, 24, 58
fixed loop nesting, 61
loop nesting, 61
metrics, 63
relative loop nesting, 62
routines and loops, 58
object files and archive libraries, 38
tasks overview, 49
instrumentor, CXoi, 38
interchange, 136
See also Optimizations
interfaces
batch mode, 6, 85–87
GUI mode, 5, 19, 50, 91
line mode, 6, 132
intrusion
behavior in, 74
event metrics, and, 47
instrumenting, and, 49
minimizing, 76
profiling strategy, 76

L

Latency, 45
Line, 119
line mode
accessing profiling data, 112
analyzing, 112–121
batch scripts, 85
command files, 85
compiling, 22
executing, 27
instrumenting, 58–64
interface, 6
invoking CXperf with a PDF, 83
shortcuts, 28
line numbers in source, 119
linker, specify path name, 38
linking object files, 35
`list` command, 116, 133
`list selectable` command, 117, 133
loop blocking, 136
loop interchange, 136
See also Optimizations
loop Latency, 136
loop nesting
minimizing profiling intrusion, 74
selecting
GUI mode, 53
line mode, 61
`set visibility` command, 61
loop slices (sections), 54
loops, 51, 58, 114
compiling and instrumenting for, 51
intrusion when profiling, 74
nesting levels, 53, 61
parallel, 58
Sort Criteria, 95
Subset Selection, 95

M

Memory events
Cache misses, 45
Data TLB misses, 45, 136
Instruction TLB misses, 45, 136
Latency, 45, 136
merging PDFs, 80
Message Passing Interface
See MPI
Metric selection button, 99
metrics
analyzing
GUI mode, 98–109
line mode, 112–121, 131–133
architecture-dependent, 44
architecture-independent, 43
CXperf, available in, 9
Data and Instruction TLB misses
Data TLB misses, 46
Instruction counts, 46
Instruction TLB misses, 46
Data Cache Utilization
Cache misses, 45

Instruction TLB misses, 45
Latency, 45
default, 43
derived, 46
instrumenting
GUI mode, 56
line mode, 63
Memory events
Cache misses, 45
Data TLB misses, 45
Instruction counts, 45
Instruction TLB misses, 45
Latency, 45
minimizing intrusion, and, 47
overview, 42
performance, and, 9, 42
Process events
Context switches, 44
Migrations, 44
Page faults, 44
Timer
CPU time, 43
CPU/Wall Clock, 43
Execution counts, 43
Wall Clock, 43
Migration, 44, 136
millicode functions
See CXoi limitations
millions instructions per second
See MIPS
minimizing intrusion, 76
MIPS, 47, 137
See also derived metrics
MPI
definition, 137
merging PDFs, 80
profiling applications, 79
multi-process applications, 79

N

naming PDFs, 69
Nesting level, 120
See loop nesting
nodes, in Call Graphs, 109

O

object files
instrumenting and linking, 36
instrumenting with CXoi, 38
Optimization level, 137
Optimizations, 126, 137
optimized loops, 120
Opts
See Optimizations

P

Page faults, 44, 137
PAGER environment variable, 28, 114, 131
parallel concurrency, 127
parallel loops, 51, 58, 114
Parallel Profile, 92, 106
interpreting data for merged PDFs, 82
overview, 7
rotation in graph, 107
Parallel Report, 92, 122–127
Loop Performance Analysis, 125
Parallel Loop Performance Analysis, 127
Routine Performance Analysis, 125
parallel scaling, 127
Parallel Virtual Machine
See PVM
parent processes (routines), 77, 104
Pause button, 19
PDF *See* Performance Data Files
Performance Data Files, 69, 83
analyzing
graphically, 100–109
GUI mode, 68
line mode, 70
multiple PDFs, 93
text reports, 110–133
changing directory path for , 80
changing during a CXperf session, 83, 115
generating, 68
generating for MPI and PVM applications, 79
invoking CXperf with, 68, 69, 70, 83
location of, 66
naming convention, 68, 79
preventing overwriting, 83, 115
PROFDIR environment variable, 66, 80
reports
GUI configurations, 95–99
line mode configurations, 112–119
selecting
in GUI mode, 101
line mode, 112, 115
Tear Off Analysis, 93
performance reports
graphical, 7, 100–109
line mode, 112–121, 131
overview, 9
text, 9
GUI mode, 110, 122–130
line mode, 112, 131
plus children, 120
Preinstrument Executable, 57, 67
preinstrumented executable file
running, 67
saving
in GUI mode, 67
in line mode, 69
preinstrumenting, 57, 65
creating PDFs, 65
environment settings, 65

for different architectures, 66
in GUI mode, 67
in line mode, 69
MPI and PVM applications, 65, 79
`save executable` command, 69
Save Profile, 93, 94
primary routines, 119, 129
primitive metrics *See* metrics
Process events
Context switch, 44
Context switches, 135
Migrations, 44, 136
Page faults, 44, 137
Process Identification number (PID), 99, 137
process states
during analysis, 121
during execution, 19
`prof`, 2
PROFDIR environment variable, 66, 80
profiling
critical region types, 76
data sampling points for, 74
GUI mode
analyzing, 20, 90–109, 122–131
compiling, 14
executing, 19, 100
instrumenting, 15, 49–58
overview profiling session, 12
intrusion and overhead during, 74
line mode
analyzing, 28, 112–121, 131–133
compiling, 22
executing, 27, 79, 85, 112
instrumenting, 24, 58–64
overview profiling session, 22
merged PDFs, 80
MPI and PVM applications, 79
multi-process applications, 79
time delays in, 74
using CXoi, 39
profiling intrusion
behavior in, 74
event metrics, and, 47
instrumenting, and, 49
minimizing, 76
Profiling Status, 121
profiling strategy, 2, 76
critical region types, 76
intrusion, 74
overview, 74
selecting metrics judiciously, 47, 76
selecting region types, 74
uninstrumented routines, 78
PS *See* Profiling Status
PVM
definition, 137
merging PDFs, 80
profiling applications, 79

R

Reduced Instruction Set Computer (RISC), 137
Region Detail, 103, 107
Region Detail dialog, 103, 104
Region Type button, 95
region types, 114
annotations in source, 105
compiler-generated loops, 51, 58
critical, 76
default selections
GUI mode, 51
line mode, 58
Find Region, 92, 93
loops, 51, 58
minimizing profiling intrusion, and, 74
parallel loops, 51, 58
Region Detail, 103, 104, 107
routines, 51, 58
viewing source
GUI mode, 103–104
line mode, 133
relative loop nesting, 55, 61
reports
configuration options
GUI mode, 95–99
line mode, 112–119
exclusive data, 98
fields in, 119
graphical, 90–109
inclusive data, 98
performance
graphical, 7, 100–109
overview, 9
textual, 9, 110–133
selecting in GUI mode, 92
Reset Graph button, 103, 107
RISC
See Reduced Instruction Set Computer
routines, 51, 58, 114
child, 77, 104
compiling and instrumenting for, 51
parent, 77, 104
primary, 119, 129
Sort Criteria, 95
Subset Selection, 95
uninstrumented, 77
`run` command, 27

S

Save Profile, 93, 94
`select` command, 24, 58, 63
Select Metric button, 99
selecting
in GUI mode

All/None button, 15
executable file to profile, 33
loop nesting, 17, 53
metrics to collect, 18, 56
region types to analyze, 95–97
regions to profile, 15, 49–53
in line mode
loop nesting, 61
metrics to analyze, 113–118
metrics to collect, 25, 28, 63
regions at specific lines, 60
regions to analyze, 113–118
regions to profile, 24, 58
specific routines, 59
`set events` command, 25, 63
`set pdf` command, 115
`set visibility` command, 61, 116
shared libraries
See CXoi limitations
shell script
See batch mode
shortcuts, command line, 28
Show All button, 103, 107
Show in Graph button, 103
Show in Source button, 103
Sort Criteria button, 95
Sort Criteria dialog, 96
source code
annotations in, 105, 133
viewing
GUI mode, 103–104
line mode, 133
window, 105
source code regions, 51
compiler-generated loops, 51, 58
loops, 51, 58
parallel loops, 58
routines, 51, 58
See also region types
Source Window, 105
starting CXperf
in batch mode, 85
in GUI mode, 14
in line mode, 24
with a PDF, 83
static routines, profiling
See CXoi limitations
statistical sampling, 2
status
process, during execution, 19
process, in report fields, 121
strip mining, 137
See also parallel loops
Subset Selection button, 95
Subset Selection dialog, 97
Summary Profile, 92, 102
interpreting data for merged PDFs, 82
overview, 7
Summary Report, 92, 122–127
Loop Performance Analysis, 125
Parallel Loop Performance Analysis, 127
Routine Performance Analysis, 125

T

Tear Off Analysis, 93
Text Reports
fields, 119
GUI mode, 110, 122–131
line mode, 112–121, 131–133
Thread Identification number (TID), 99, 137
threads, 137
Data Source button, 98
in performance reports, 122
TID *See* Thread Identification number
Timer metrics
Call Graph, 43
CPU time, 43, 135
CPU/Wall, 43, 135
Execution counts, 43
Wall Clock, 43, 137
timing collection routines
See cxperfmon.o
TLB *See* Translation Lookaside Buffer (TLB)
TMPDIR environment variable, 40
Toolbar, 92
Translation Lookaside Buffer (TLB), 46, 137
tty mode
shortcuts, 28
See line mode

U

uninstrumented routines, 77

V

viewing source
GUI mode, 103–104
line mode, 133

W

Wall Clock, 43, 137

Z

Zoom, 103, 107